SOME PARALLEL ALGORITHMS IN COMPUTATIONAL GEOMETRY

A Thesis Submitted

in Partial ulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

By

V PADMAVATHI

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY 1991

CERTIFICATE

This is to certify that the Thesis work entitled **Some Parallel algorithms in Computational Geometry** has been carried out under my supervision and has not been submitted elsewhere for the award of a degree

Station KANPUR

Date 4 th Feb 91

(Dr Asish Mukhopadhyay)
Assistant Professor
Dept of C S E
I I T Kanpur

CERTIFICATE

This is to certify that the Thesis work entitled **Some Parallel algorithms in Computational Geometry** has been carried out under my supervision and has not been submitted elsewhere for the award of a degree

Station KANPUR

Date 4 th Feb 91

(Dr Asish Mukhopadhyay
Assistant Professor
Dept of C S E
I I T Kanpur

ABSTRACT

Computational Geometry addresses algorithmic problems in diverse areas It is seldom obvious how to generate parallel algorithms in this area since most popular techniques involve an essentially sequential approach

Here parallel algorithms have been proposed for certain problems in Computational Geometry using CREW PRAM processors Among these are the linear programming problem in two dimensions the minimum spanning circle problem for a set of points in the plane for a set of line in the plane and also for a set of planar line segments Parallel algorithms also have been designed for the computation of the shortest line segment from which a given conve polygon is weakly e ternally visible and for the reconstruction of an orthogonal polygon given a circular embedding of its visibility graph

Acknowledgements

I would like to place on record my deep sense of gratitude to Prof A ish Muhopadhyay for his guidance e treme patience ard encouragement throughout the project I am e tremely thankful to him for his cooperation I an grateful to him for giving me enormous amount of freedom

I sincerely acknowledge Mr Srinivasa Raghavan for his suggestions and for sparing his valuable time for discussions

I thank all the members of the faculty for good instruction Their dedication to their profession and eal for the subject has been a constant inspiration that relindled my flagging interest in further studies

I am indebted to Vidya for her constant encouragement active help in case of difficulties and doubts and for her association which had been of immense educative and entertainment value I express boundless gratitude towards my friends Sujata and Geeta for their understanding nature jovial company and friendship which is highly valued by me I thank Ms Shirin and other friends for their help and company

V Padmavathi

Table of Contents

List of Figures

CHAPTER 1

INTRODUCTION

1 1 Computational Geometry and Parallel Algorithms

Computational Geometry addresses algorithmic problems in diverse areas A large number of application areas such as pattern recognition computer graphics image processing operations research statistics computer aided design robotics etc have been the incubation bed of this discipline since they provide inherently geometric problems for which efficient algorithms have to be developed This field has been flourishing because of the rich interaction with many application areas

Visibility is one of the most fundamental topics in Computational Geometry Visibility problems find application in many areas such as graphics automated cartography image processing and robotics Also visibility problems often appear as subproblems of many other problems in computational geometry (like the shortest path problems with obstacles)

Since 1975 there has been a rapid development of sequential algorithms for geometric problems but until 1985 there was little published about parallel algorithms for such problems

It is seldom obvious how to generate parallel algorithms in this area since popular techniques include an explicitly sequential (iterative) approach

Here parallel algorithms have been developed for problems in visibility and optimisation

1 2 Survey

Serial Algorithms

There has been a wide development of sequential algorithms for geometric problems We report here a few results that are particularly relevant to our work

The notion of weak visibility has received attention in both mathematics and computer science literature Avis and Toussaint [AT81] showed that given a simple polygon P and a specified edge e whether P is edge visible from e can be determined in O(n) time Sack and Suri [SS89] discovered a linear time algorithm for determining all such edges of a given polygon Bhattacharya [BT89] gave a linear time algorithm for computing the shortest line segment from which a given conve polygon is weakly externally visible

A linear time algorithm has been given in [OJ87] for the reconstruction of an orthogonal polygon given a circular embedding of its visibility graph

Nimrod Meggido [MN83] gave a linear time algorithm for linear programming in R^2 and R^3

Parallel Algorithms

Not much has been published on parallel algorithms for geometric problems Some of the results in parallel algorithms in Computational Geometry are discussed below

Atallah Cole and Goodrich showed that one can compute the portion of the plane visible from a point of the plane in the presence of a collection of n line segments in O(log n) time using O(n) processors The number of processors needed for this problem was improved by Atallah and Chen [AC89] to O(n/log n) for the case when the line segments form a simple polygon ElGindy and Goodrich [EG87] showed that one can determine the shortest path between two points in a simple polygon in O(log n) time using O(n) processors and one can construct the shortest path tree from a verte v (the union of all shortest paths from v to other vertices) in O(log n) time using O(n) processors Aggarwal Chagelle Guibas et al [AC85] presented efficient parallel algorithms for basic problems in Computational Geometry vi convex hulls voronoi diagrams detecting line segment intersections triangulating simple polygons minimising a circumscribing triangle in O(log n) O(log n) O(log n) O(log n) time with O(n) O(n) O(n log n) O(n log n) O(n) processors respectively Recently Goodrich Shauck Sumanta Guha [GS90] showed that a data structure called the stratified decomposition tree can be built in O(log n) time using O(n) processors that allows one to construct an implicit representation of the shortest path between two points inside P in O(log n) time using O(n) processors It is also shown that the visibility graph from the vertices of P can be computed in O(log n) time using O(n log n + I/log n) processors where I is the number of edges in the graph

For all these algorithms the model of computation is the

CREW PRAM model (Concurrent Read E>clusive Wr1te Parallel RAM)

1 3 Model of Computation

An SIMD (Single Instruction Multiple Data) CREW (Concurrent Read E clusive Wr1te) PRAM (Parallel RAM) model of computation is used An SIMD computer consists of a number of processors operating under the control of a single instruction stream issued by a central control unit The processors each have a small private memory for storing programs and data and operating synchronously during a given time unit a selected number of processors are active and e ecute the same instruction each on a different data set the remaining processors are inactive The processors are synchronised in the sense that if a set of instructions is e ecuted in parallel then each must be allowed to finish before the ne>t set of instructions is started

In this model there is an unbounded global memory which is shared by all the processors Processors communicate with each other through this common memory No two processors are allowed to write on the same location of the global memory simultaneously But any number of proces_ors can simultaneously read from the same location of the global memory

E>ecution of Parallel Algorithms A parallel algorithm is one that is designed to run on a parallel computer

Parallel running time is the most important measure in evaluating a parallel algorithm It is defined as the time required to solve a problem that is the time elapsed from the

moment the algorithm starts to the moment the algorithm terminates For a problem of size n the parallel worst case running time of an algorithm a function of n is denoted by t(n)

A good indication of the quality of a parallel algorithm for some problem is the speedup it produces

$$\text{speedup} = \frac{\text{Worst case running time of the fastest known sequential algorithm for the problem}}{\text{Worst case running time of the parallel algorithm}}$$

The number of processors required to solve a problem is another criterion for assessing the value of a parallel algorithm The number of processors required by an algorithm a function of n is denoted by p(n)

The cost of a parallel algorithm is defined as the product of the parallel running time and the number of processors used For a problem of size n the cost of the parallel algorithm a function of n will be denoted by c(n)

$c(n) = p(n) * t(n)$

The efficiency of a parallel algorithm is defined as

$$\text{efficiency} = \frac{\text{Worst case running time of fastest known sequential algorithm for the problem}}{\text{Cost of the parallel algorithm}}$$

One of the major tasks of parallel algorithm design is to come up with parallel algorithms that are optimal i e that run as fast as theoretically possible for the problem they solve and simultaneously have a time * processors bound that is within a constant factor of the time complexity of the best known

sequential algorithm for the problem

1.4 Overview of the thesis

In chapter 2 parallel algorithms have been proposed for the Linear Programming Problem and related problems

In chapter 3 a parallel algorithm is given for the computation of the shortest line segment from which a given convex polygon is weakly externally visible

In chapter 4 a parallel algorithm has been presented for the reconstruction of an orthogonal polygon given the circular embedding of its visibility trees

We conclude in chapter 5 summarising the results obtained and indicating directions in which further research can be carried out

CHAPTER 2

LPP AND RELATED PROBLEMS

2.1 Introduction

Several known problems in computational geometry are closely related to the problem of linear programming. The problem of finding the smallest circle enclosing n given points in the plane, the problem of computing the minimum spanning circle of a set of n given lines, line segments are all related to the problem of linear programming.

Nimrod Megiddo proposed a linear time algorithm for linear programming in R^2 and R^3 [MN83]. He also proposed linear time algorithms for the minimum spanning circle problem of a set of points. Very recently linear time algorithms for finding the minimum spanning circle of a set of lines and line segments have been proposed by Mukhopadhyay, Bhattacharya et al. [BM90] and [JM90].

In section 2.2 we give a parallel algorithm for the Linear Programming Problem in two dimensions. In section 2.3 the minimum spanning circle problem for a set of points is solved. In sections 2.4 and 2.5 the problems of computing the minimum spanning circle for a set of lines and line segments are dealt. All the algorithms run in $O(n^e)$ time on $O(n^{1-e})$ CREW PRAM processors.

2 2 The Linear Programming Problem

2 2 1 The Problem

The linear programming problem in two dimensions can be stated as

$$\min_{(x_1\ x_2)} \quad c_1 x_1 + c_2 x_2$$

such that $a_{11} x_1 + a_{12} x_2 \geq \beta_1 \qquad (i = 1 \quad n)$

It is convenient for us to deal with the problem in an equivalent form

$$\min_{(x\ y)} \quad y$$

such that $y \geq a_i + b_i \quad (i \in I_1)$

$y \leq a_i + b_i \quad (i \in I_2) \qquad a \leq \quad \leq b$

where $|I_1| + |I_2| \leq n \qquad \infty \leq a \quad b \leq \infty$

Define the functions

$$g(\ ) = \max \{ a_i + b_i \quad i \in I_1 \}$$

$$h(\ ) = \min \{ a_i + b_i \quad i \in I_2 \}$$

Both these functions are piece wise linear and g is conve while h is concave A number x a ≤ ≤ b is said to be feasible if g( ) ≤ h(x) The problem can be posed in a one dimensional form

minimise g( )

such that g( ) ≤ h(y)

a ≤ x ≤ b

Testing a value of x Given any value x of x (a ≤ x ≤ b) we test (i) Is x feasible (ii) If x is not feasible then if there are any feasible values of x then they must all lie on one side of

x our test either determines that side or concludes that no feasible value exists (iii) If y is feasible then our test will recognise whether is also optimal and if not it will tell us on what side of x the minimum lies (see fig 2 2 1)

Define

$$s_g = \min \{a_i \quad i \in I_1 \quad a_i y + b_i = g(x)\}$$

$$S_g = \max \{a_i \quad i \in I_1 \quad a_i x + b_i = g(x)\}$$

$$s_h = \min \{a_i \quad i \in I_2 \quad a_i + b_i = h(y)\}$$

$$S_h = \max \{a_i \quad i \in I_2 \quad a_i x + b_i = h(x)\}$$

Case i g(x) h( )

Consider the function f(x) = g( ) h( )

This function is conve so all the values of x such that f( ) ≤ ∅ lie on one side of In order to tell the correct side we look at the one sided derivative of f at

If s_g S_h then f( ) is ascending at x so that a feasible can only be smaller than If S_g s_h then f( ) is descending at x so that a feasible x can only be larger than The remaining case is wher s_g $S_h \leq \emptyset \leq S_g$ s_h In this case f attains its minimum at x i e there are no feasible values of

Case ii g( ) < h(x)

If $s_g > \emptyset$ then the optimal solution (denoted by x^*) must satisfy x^* y If $S_g < \emptyset$ then x^* x Otherwise $s_g \leq \emptyset \leq S_g$ and y itself is a minimum of g

Case iii g(x) = h(x)

If s_g ∅ and $s_g \geq S_h$ then $x^* < x$

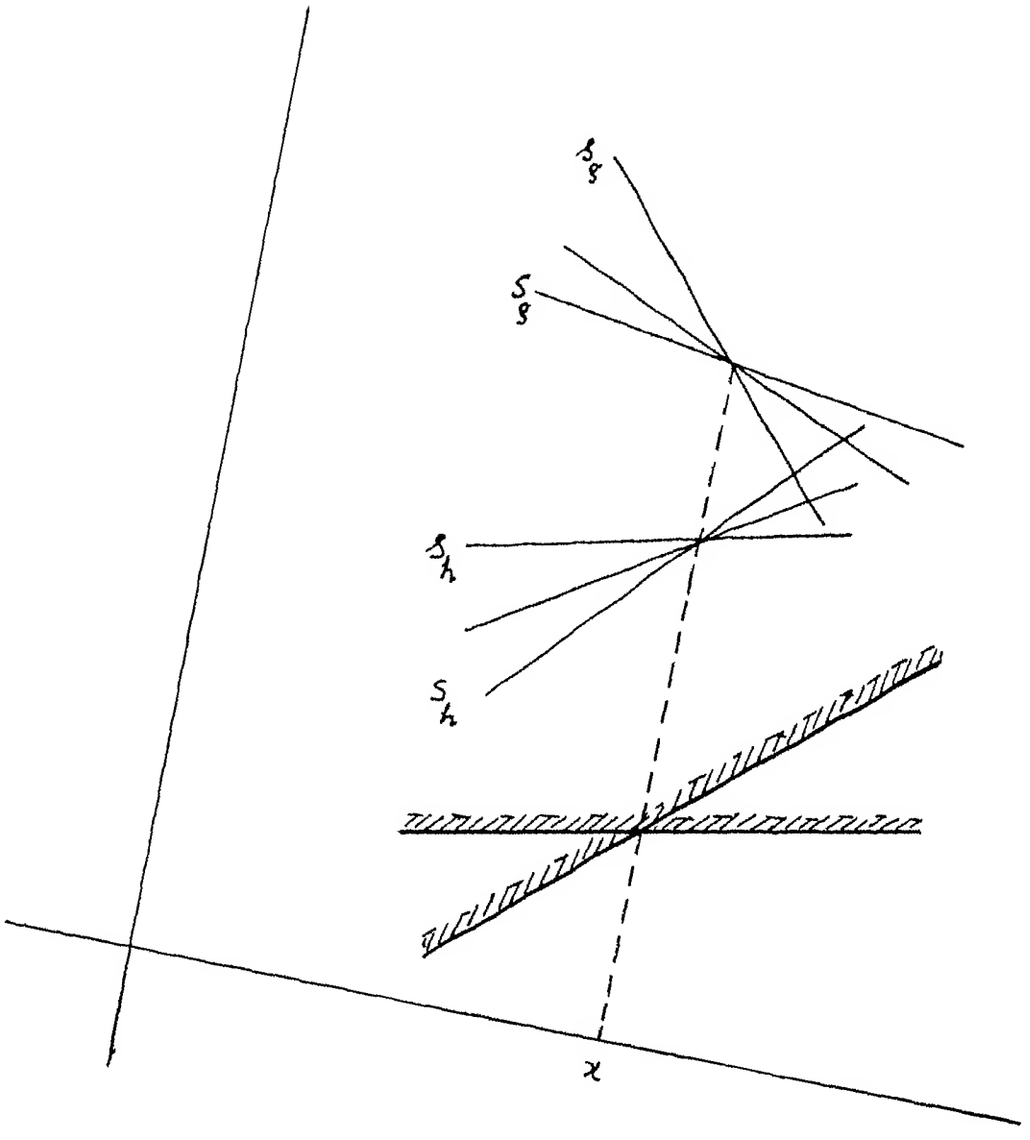


Fig 2 2 1

If $S_g \neq \emptyset$ and $S_g \leq s_h$ then $x^* = x$

Otherwise x itself is a minimum of $g(x)$ under the constraint $g(x) \leq h(x)$

2 2 2 The Algorithm

The procedure *par-lpp* solves the LPP problem in $O(n^e)$ time using $O(n^{1-e})$ CREW PRAM processors *Par-sel* is used by *par-lpp* The procedure *par-sel* gives the k^{th} element of the input sequence of length n in $O(n^e)$ time using $O(n^{1-e})$ processors [A^8^]

par-sel(S k)

(1) If $|S| \leq 3$ then using one processor return the k^{th} element

else

subdivide S into $|S|^{1-e}$ subsequences each of size $|S|^e$ and assign a subsequence to each processor

(2) For $i = 1$ to $|S|^{1-e}$ do in parallel

P_i finds the median m_i of its associated subsequence using the sequential procedure *select* P_i writes m_i in $M[i]$ the i^{th} position of the array M in shared memory

(3) Find the median m of M by calling *par-sel(M $|M|/2$)*

(4) Subdivide S into three subsequences S_1 S_2 S_3 of elements smaller than equal to and larger than m respectively

(5) If $|S_1| \geq k$ then *par-sel(S_1 k)*

else

if $|S_1| + |S_2| \geq k$ then return m

else *par-sel(S_3 $k - |S_1| - |S_2|$)*

end if

end if

Analysis

Step 1 takes constant time

Step 2 needs $c_1 n^e$ time (where c_1 is a constant) as the sequential select is of linear time

Step 3 takes $t(n^{1-e})$ time as $|M| = n^{1-e}$

Subdividing S into S_1 S_2 S_3 can be done by letting each processor P_i split its associated sequence into three lists S_1^1 S_2^1 and S_3^1 of elements smaller than equal to and larger than m respectively in $O(n^e)$ time Then step 4 takes $O(n^e)$ time The S_1^1 S_2^1 and S_3^1 are merged to form S_1 S_2 S_3 respectively as follows To merge S_1^1 denote $s_i = |S_1^1|$ For each i $_i = \Sigma s_j$ (j = 1 to n) is found by parallel prefix computation in $O(\log n^{1-e})$ time Taking $_0 = 0$ all the processors simultaneously write their lists in S_1 with P_i starting to copy S_1^1 in position $_{i-1} + 1$ of the array S_1 Hence the time required by step 4 is dominated by $c_2 n^e$ where c_2 is a constant

Since m is the median of M $n^{1-e}/2$ elements of S are guaranteed to be larger than it Every element of M is smaller than $n^e/2$ elements of S Thus $|S_1| \leq 3n/4$ Similarly $|S_3| \leq 3n/4$ Hence step 5 takes $t(3n/4)$ time

$$t(n) = c_1 n^e + t(n^{1-e}) + c_2 n^e + t(3n/4) = O(n^e)$$

The following procedure solves the LPP problem The problem is taken to be in the form

min y
(y)

such that $y \geq a_i x + b_i \quad (i \in I_1)$

$y \leq a_i x + b_i \quad (i \in I_2) \qquad a \leq x \leq b$

where $|I_1| + |I_2| \leq n \qquad -\infty \leq a \; b \leq \infty$

Arrays A B contain a_i s and b_i s respectively

par-lpp(I_1 I_2 A B a b)

(1) If $|I_1|$ and $|I_2|$ are con tant use the sequential algorithm else divide the set I_1 into subsets of si e n^e each and assign a processor to each Similarly divide the set I_2 into subsets of si e n^e and as ign a processor to each

(2) For i = 1 to n^{1-e} do in parallel

P_i arranges the elements in its associated sequen e in di joint pairs

$X_i = \{\}$

For each pair (j j') in its associated sequence P_i does

If $a_j = a_{j'}$ then if (j j') $\in I_1$ then drop one of the constraints

$y \geq a_{j'} + b_{j'}$ and

$y \geq a_j + b_j$

else drop one of the constraints

$y \leq a_{j'} x + b_{j'}$

$y \leq a_j x + b_j$

else $x_{j,j'} = (b_j - b_{j'}) / (a_{j'} - a_j)$

If $x_{j,j'} \notin [a \; b]$ then drop one of the constraints

else add y_{ij} to the set X_i

(3) Let X = union of all X_i s

y_π = *par sel(X |X|/2)*

(4) Let C = { } H = { }

For i = 1 to n^{1-e} do in parallel

P_i finds $f_{ij} = a_j y_\pi + b_j$ for each $j \in$ the associated subsequence of P_i

If the associated subsequence of $P_i \in I_i$ then P_i adds f_{ij} to the et G

else adds to H

(5) g = *par-sel(G |G|)*

h = *par-sel(H 1)*

(6) Let S1 = { }

S = { }

For i = 1 to n^{1-e} do in parallel

For each element j in its associated subsequence P_i does the following

if $a_j y_\pi + b_j$ = g then add a_j to S1

if $a_j y_\pi + b_j$ = h then add a_j to S2

(7) s_g = *par-sel(S1 1)*

s_h = *par-sel(S2 1)*

S_g = *par-sel(S1 |S1|)*

S_h = *par-sel(S2 |S2|)*

(8) P_i checks g and h as follows

case

g h if $s_g > \emptyset$ then [a b] = [a $\rangle_\pi$]

if $S_g < \emptyset$ then [a b] = [$\rangle_\pi$ b]

else return $\rangle_\pi$

g = h if $s_g \succ \emptyset$ and $s_g \geq S_h$ then [a b] = [a $\succ_\pi$]

if S_g $\emptyset$ and $S_g \leq s_h$ then [a b] = [$\rangle_\pi$ b]

else return $\rangle_\pi$

g h if s_g S_h then [a b] = [a $_\pi$]

if S_g s_h then [a b] = [$_\pi$ b]

else return infeasible

(9) For i = 1 to n^{1-e} do in parallel

For each element $\rangle \in X_1$ P_1 tests if $\rangle \in$ [a b]

if $\notin$ [a b] then drop one of the constraints

end for

end for

Let I_1 I_2 be the sets obtained after dropping the constraints

(10) *par-lpp*$\langle I_1$ I_2 A B a $b\rangle$

2 2 3 Analysis

Steps 2 4 6 and 9 require $O(n^e)$ time as each processor spends a linear amount of time on its input

Steps 3 5 7 take $O(n^e)$ time since this is the time taken by *par-sel*

Here the formation of the sets X H G I_1 I_2 can be done in the lines similar to the one used to form S_1 in *par-sel* (as described in the analysis of step 4 of *par-sel*) and hence can be

dere in $O(n^e)$ time

At least half of the critical values x_{ij} will not be in the interior of the new interval We will thus be able to drop one constraint per each such pair which is at least a quarter of the set of constraints (including those that have been dropped prior to the evaluation of x_π) We are thus left with a LPP in the plane with atmost 3n/4 constraints Hence step 10 takes t(3n/4) time since after elimination 3n/4 constraints remain

Rest of the steps take constant time

$$t(n) = t(3n/4) + c_1 n^e = O(n^e)$$

2 3 Minimum spanning circle for a set of points

2 3 1 The Problem

Given a set of n points $(a_i \quad b_i)$ i = 1 to n in the Euclidean plane the problem is to find the smallest circle enclosing these points Formally we are looking for a point (y) so as to minimise

$$\max \{ ((x - a_i)^2 + (y - b_i)^2)^{1/2} \quad 1 \le i \le n \}$$

Thus the point (y) is an optimal location of a facility if we wish to minimise the largest distance that a customer would have to travel from his residence to the facility

We first develop a parallel algorithm for the constrained case where the center is constrained to lie on the x axis Consider the problem of minimising

$$g(x) = \max_i \{ (x - a_i)^2 + b_i^2 \quad 1 \le i \le n \}$$

2 3 2 The Algorithm

Let x^* denote the minimiser of $g(x)$

The procedure *par-cons-point* determines the center and the radius of the minimum spanning circle (constrained case) of the given n points using $O(n^{1-e})$ CREW PRAM processors in $O(n^e)$ time

par-cons-point(S)

(1) If |S| is not a constant subdivide S into subsequences of size n^e and assign a processor to each subsequence

(2) X = { }

For i = 1 to n^{1-e} do in parallel

P_i pairs up the points in its associated subsequence

For each pair $(a_j \quad a_{j+1})$ in its associated subsequence P_i does

if $a_j = a_{j+1}$ drop one of the points $(a_j \quad b_j)$ and $(a_{j+1} \quad b_{j+1})$

compute $x_{j\,j+1} \quad (a_{j+1}^2 - a_j^2 + b_{j+1}^2 - b_j^2) / 2(a_{j+1} - a_j)$

add $x_{j\,j+1}$ to X

end for

end for

(3) x_m = *par-sel(X |X|/2)*

(4) F = { }

For i = 1 to n^{1-e} do in parallel

P_i computes $f_j = (x_m - a_j)^2 + b_j^2$ for each j in its associated subsequence and adds it to F

end for

(5) g m = *par-sel(F |F|)*

(6) I = { }

For i = 1 to n^{1-e} do in parallel

For each point (a_j, b_j) in its associated subsequence P_i does

if $(>_\pi a_j)^2 + b_j^2 = g_\pi$ add j to I

end for

end for

(7) Initialise right = false left = false center = false

If |I| n^e use one processor

else

Subdivide I into subsequences of size n^e and assign each of these subsequences to a processor

For i = 1 to n^{1-e} do in parallel

P_i initialises $right_i$ = false $left_i$ = false $centre_i$ = false

sets $right_i$ = true if $>_\pi a_j$ for every j ∈ its associated subsequence

sets $left_i$ = true if $>_\pi > a_j$ for every j ∈ its associated subsequence

end for

(8) Set left = true if all $left_i$ = true

Set right = true if all $right_i$ = true

else return $(>_\pi \emptyset)$ and g_m

(9) For i = 1 to n^{1-e} do in parallel

P_i does

if left = true then if $_{j\ j+1} >_\pi$ discard (a_j, b_j)

if right = true then if $x_{J\ J+1} < y_{\pi}$ discard $(a_{J+1}\ b_{J+1})$

end for

S - set of remaining points

(10) *par-cons-point(S)*

Analysis

Step 1 takes constant time

Steps 2 4 6 7 and 9 take $O(n^e)$ time as in each of the e each processor spends a linear amount of time on its input

Steps 3 and 5 take $O(n^e)$ time as they use *par-sel*

Step 8 takes $O(n^e)$ time This can be done by the method of parallel prefix computation

The formation of sets X F I can be done in lines similar to the one used to form S_1 in the *par-sel* (as described in the analysis of step 4 of *par-sel*) and hence can be done in $O(n^e)$ time

We discard all together in steps 2 and 9 atleast one quarter of the points Therefore step 10 takes a time of t(3n/4)

$$t(n) = t(3n/4) + c_1 n^e = O(n^e)$$

We now address the question of recognising on what side of the straight line the unconstrained center lies The function $f(x\ y) = \max\{(x - a_1)^2 + (y - b_1)^2 \quad 1 \le i \le n\}$ is convex f is convex not only in each variable but also as a function of two variables

$h(y) = \min_x f(x\ y)$ is convex

By minimising g( ) we evaluate h(0) The y coordinate of the

unconstrained center y^c is precisely where $h(y)$ attains its minimum. Since $h(y)$ is convex we can tell the sign of y^c simply by looking in the neighbourhood of $y = \emptyset$. Let $(x^*, \emptyset)$ be the center in the constrained case. Let

$$I = \{ i \mid (x^* - a_i)^2 + b_i^2 = g(x^*) \}$$

If $I = \{i\}$ then $x^* = a_i$ and y^c has the sign of b_i. If $I = \{ i, j \}$ then $(x^*, \emptyset)$ lies on the perpendicular bisector of the line segment $[(a_i, b_i), (a_j, b_j)]$. y^c has the sign of the y coordinate of the midpoint of this segment, i.e. $1/2(b_i + b_j)$. In general all the points (a_i, b_i), $i \in I$ lie on a circle centered at $(x^*, \emptyset)$. If $(x^*, \emptyset)$ is in the convex hull of these points then $y^c = \emptyset$. Otherwise there exist two points (a_i, b_i), (a_j, b_j), $(i, j \in I)$ such that $f(x, y)$ decreases as we move from $(x^*, \emptyset)$ in the direction of the midpoint of the line segment $[(a_i, b_i), (a_j, b_j)]$ (i e. along the perpendicular bisector of that segment). In this case y^c has the sign of $(b_i + b_j)/2$. The determination of these two points can be carried out in $O(n^e)$ time using $O(n^{1-e})$ processors as follows.

Apply affine transformations to transform $(x^*, \emptyset)$ and (a_i, b_i) to $(\emptyset, \emptyset)$ and $(\emptyset, 1)$ respectively. If $(\emptyset, \emptyset)$ is not in the convex hull of the given points then $\exists$ an α such that $b_i \leq \alpha a_i$ for every $i \in I$. i.e. α must satisfy

$$\max \{ b_i/a_i \mid a_i > \emptyset \} \leq \alpha \leq \min \{b_i/a_i \mid a_i < \emptyset \}$$

and also $b_i \geq \emptyset$ for every i such that $a_i = \emptyset$. If this is the case the points needed by us are the points at which the maximum

ard the minimum are obtained on the LHS and RHS respectively Note that this maximum and the minimum points can be obtained easily using $O(n^{1-e})$ processors in $O(n^{e})$ time If (a_1, b_1) (a_J, b_J) are the points so found then the sign of y^c is that of $1/2(b_1 + b_J)$ Otherwise $y^c = 0$

Thus we can determine on what side of the line the center of the smallest enclosing circle lies in $O(n^{e})$ time using $O(n^{1-e})$ processors If the center lies on this line then we discover its exact location during the procedure

Algorithm for the unconstrained case

The following procedure solves the minimum spanning circle problem using $O(n^{1-e})$ CREW PRAM processors in $O(n^{e})$ time

par-uncons-points(S)

(1) If |S| is a constant number use the sequential algorithm Else subdivide the set of input points into subsequences of size n^{e} and assign a processor to each subsequence

For i = 1 to n^{1-e} do in parallel

P_i pairs up the points in its associated subsequence and finds the bisectors l_j s of the line segments between the pairs of points

end for

(2) Let L be the set of all the bisectors found above

Divide the lines into two sets L_1 L_2 with the lines making angles α β with the x axis where $0 \leq \alpha \leq \pi/2$ and $\pi/2 \leq \beta \leq 0$

(3) Find the line of median slope (angle α_m with the x axis) among the lines in the set L_1 using *par-sel* Divide L_1 in to two sets one with slopes the median slope and the other with slopes the median slope Repeat the steps upto step 9 with the set L_2 taking the median slope line of the set L_2

(4) For i = 1 to n^{1-e} do in parallel

P_i pairs up the lines in its associated subsequence so that each pair has one line with nonpositive angle and the other with nonnegative angle considering the linear transformation that takes the x axis to the line $y = \alpha_m$

For each pair (l_i, l_j)

If (l_i, l_j) are parallel to the x axis y_{kj} is the mean of the y coordinates Otherwise find (x_{kj}, y_{kj}) the point of their intersection

end for

end for

(5) Find the median of all such y_{kj} s using *par-sel*

(6) Test on which side of the line $y = y_m$ the center lies using the method outlined in the earlier section

(7) If the center lies on the line $y = y_m$ we are done

If the center lies underneath $y = y_m$

X = { }

For a = 1 to n^{1-e} do in parallel

P_a considers the pairs (l_1, l_j) in its associated set

If $l_1 \nparallel l_j$ and $y_{1j} \geq y_m$ and l_1 lies above $y = y_m$ then drop P_i

where l_i is the bisector of (p_k, p_0) and p_k lies underneath l_i

If l_i l_j are not parallel and $y_{ij} \geq y_\pi$ then add y_{ij} to the set X

end for

(8) Find the median x_π of X

(9) Determine on what side of the line $= x_\pi$ the center lies using the method outlined in the previous section

(10) If the center lies to the left of this line (the remaining cases can be dealt similarly)

For i = 1 to n^{1-e} do in parallel

P_i considers pairs (l_k, l_j) such that $y_{kj} \geq y_\pi$ and $y_{kj} \geq y_\pi$

If l_k forms a nonpositive angle with the positive direction of the x axis drop the point which lies southwest of it (fig 2.3.1)

end for

(11) Let S be the set of remaining points *par-uncons-line(S)*

2.3.3 Analysis

Step 1 4 7 10 take $O(n^e)$ time because each processor in these steps spends linear amount of time on its input

Step 2 takes $O(n^e)$ time by following the analysis of step 4 in *par-sel*

Step 3 takes $O(n^e)$ time since the median can be found in $O(n^e)$ time and the division of the set into two can be done as in step 4 of *par-sel* in $O(n^e)$ time

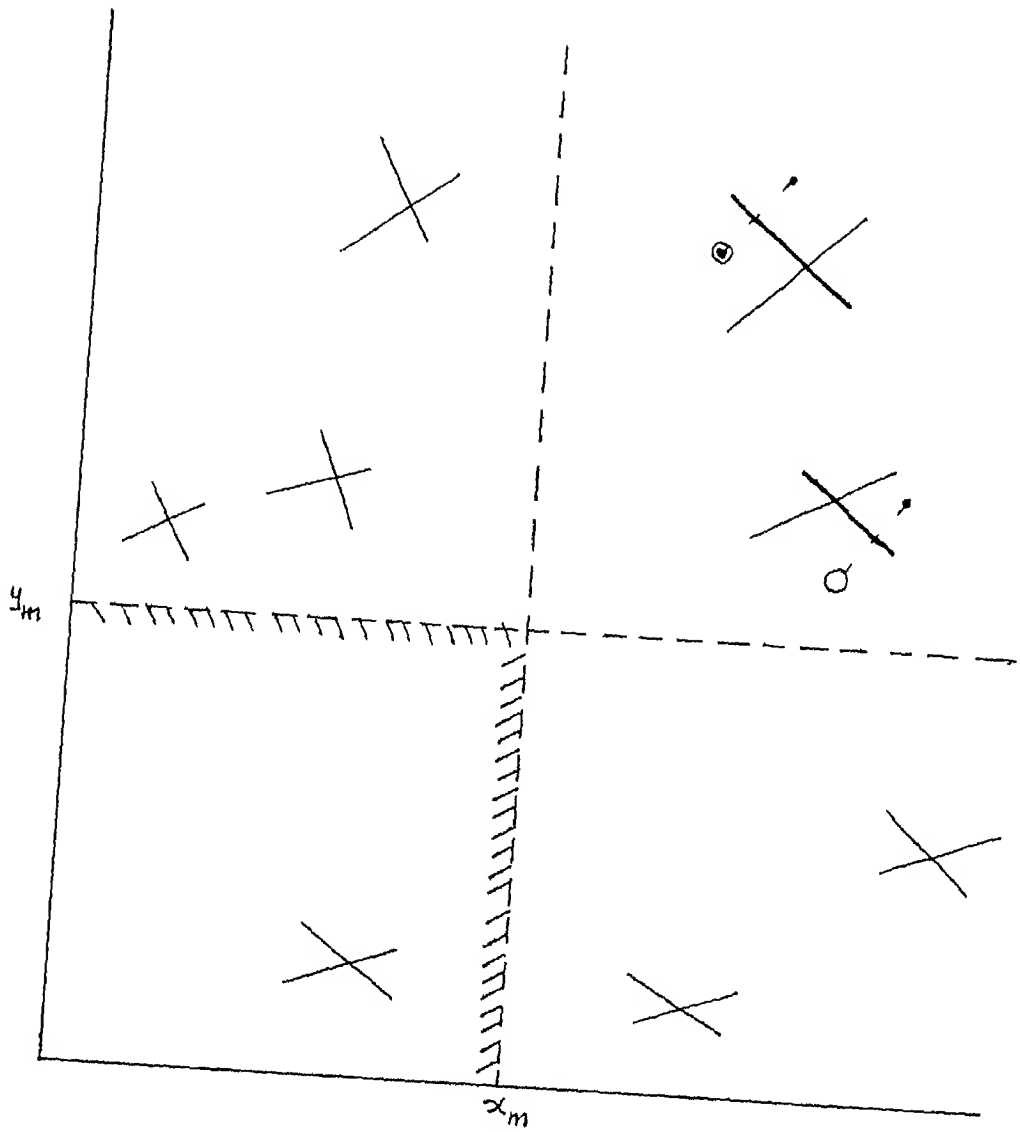


Fig 2 3 1 Circled points are to be discarded

Steps 5 and 8 take $O(n^e)$ time because they use *par-sel*

Steps 6 and 9 take $O(n^e)$ time by the method given earlier

We drop atleast one sixteenth of the points in each iteration Therefore step 11 takes $t(15n/16)$ time

$$t(n) = t(15n/16) + c_1 n^e = O(n^e)$$

2 4 Minimum spanning circle for a set of lines

2 4 1 The Problem

The problem is to compute the smallest radius circle which intersects all the given n lines in the plane The radius of such a circle is called the intersection radius A sequential linear time algorithm for this problem has been proposed in [BM90]

Let $l_i = a_i x + b_i y + c_i = 0$ $i = 1, 2, \ldots, n$ be the given set of lines in the plane The bisectors of their two lines l_i l_j $i \neq j$ are given by

$$\frac{(a_i + b_i y + c_i)}{(a_i^2 + b_i^2)^{0.5}} = \frac{(a_j + b_j y + c_j)}{(a_i^2 + b_i^2)^{0.5}}$$

We denote these bisectors by b_{ij}^1 and b_{ij}^2

2 4 2 The Algorithm

We first consider the constrained case where the center of the circle is constrained to lie on a given line L We let this line to be the x axis The following procedure computes the center of the minimum spanning circle for a set of lines using $O(n^{1-e})$ processors The input to the procedure is a set of lines S and a

line L (here the a is) The output is the center of the circle

par-cons-line(S)

(1) If |S| is a constant use the sequential algorithm else subdivide the input set of lines into subsequences of size n^e and assign a processor to each sequence

(2) For i = 1 to n^{1-e} do in parallel

P_i finds the intersection points of these lines in its associated subsequence with the x axis

end for

(3) Let X be the set of the coordinates of the points found in step 2 above

x_m = *par-sel(X, |X|/2)*

(4) D = { }

For i = 1 to n^{1-e} do in parallel

P_i finds the distance d_{ij} = distance of (x_m, 0) from L_j for each L_j ∈ the set of lines associated to P_i and adds d_{ij} to D

end for

(5) g = *par-sel(D, |D|)*

(6) I = { }

For i = 1 to n^{1-e} do in parallel

P_i adds j to I if dist((x_m, 0), L_j) = g

end for

(7) Initialise right = false left = false center = false

If |I| n^e use one processor

else

Subdivide I into subsequences of size n^{e} and assign each of these subsequences to a processor

For $i = 1$ to n^{1-e} do in parallel

P_i initialises $right_i$ = false $left_i$ = false $centre_i$ = false

P_i sets $right_i$ = true if every $l_j \in$ its associated subsequence touches the right semi-circle (utv in fig 2 4 1) centered at (π 0) and sets $left_i$ = true if they touch the left semi circle (usv in fig 2 4 1)

end for

(8) Set left = true if all $left_i$ true

Set right = true if all $right_i$ = true

else return (π 0)

(9) F = { } Y = { }

For $i = 1$ to n^{1-e} do in parallel

For each l_j in its associated subsequence P_i does

if right = true then if l_j intersects a is to the left of (π 0) add l_j to F

if left = true then if l_j intersects a is to the right of (π 0) add l_j to T

end for

end for

(10) Subdivide F into subsequences of size n^{e} and assign a processor to each sequence (Repeat the steps with the set T)

For i = 1 to n^{1-e} do in parallel

P_i pairs up l_j s in F as (l_{2j-1}, l_{2j}) such that l_{2j-1} intersects the x-axis to the left of the intersection of l_{2j} with the x-axis Let a_j be the intersection of l_j with the x-axis For each pair (l_{2j-1}, l_{2j}) add the bisector d_j whose intersection with the x-axis doesn't lie between a_{2j-1} and a_{2j} to the set D (fig 2.4.2)

end for

(11) Find d_m the median of the intersections of d_j s with the x-axis using *par-sel*

(12) Determine which side of $(d_m, 0)$ the center $(x^*, 0)$ of the required circle lies as in steps 7 and 8

(13) For i = 1 to n^{1-e} do in parallel

For each $d_j \in$ its associated subsequence P_i does

if $x^* < d_m$ then if d_j intersects x-axis to the left of d_m discard l_{2j-1} (fig 2.4.2)

if $x^* > d_m$ then if d_j intersects x-axis to the right of d_m discard l_{2j} (fig 2.4.3)

end for

(14) Let S = set of remaining lines *par-cons-line(S)*

Analysis

Let t(n) be the time for *par-cons-line*

Step 1 takes constant time

Steps 2, 4, 6, 7, 9, 10 and 13 take $O(n^e)$ time as each processor has to spend a linear amount of time on its input for

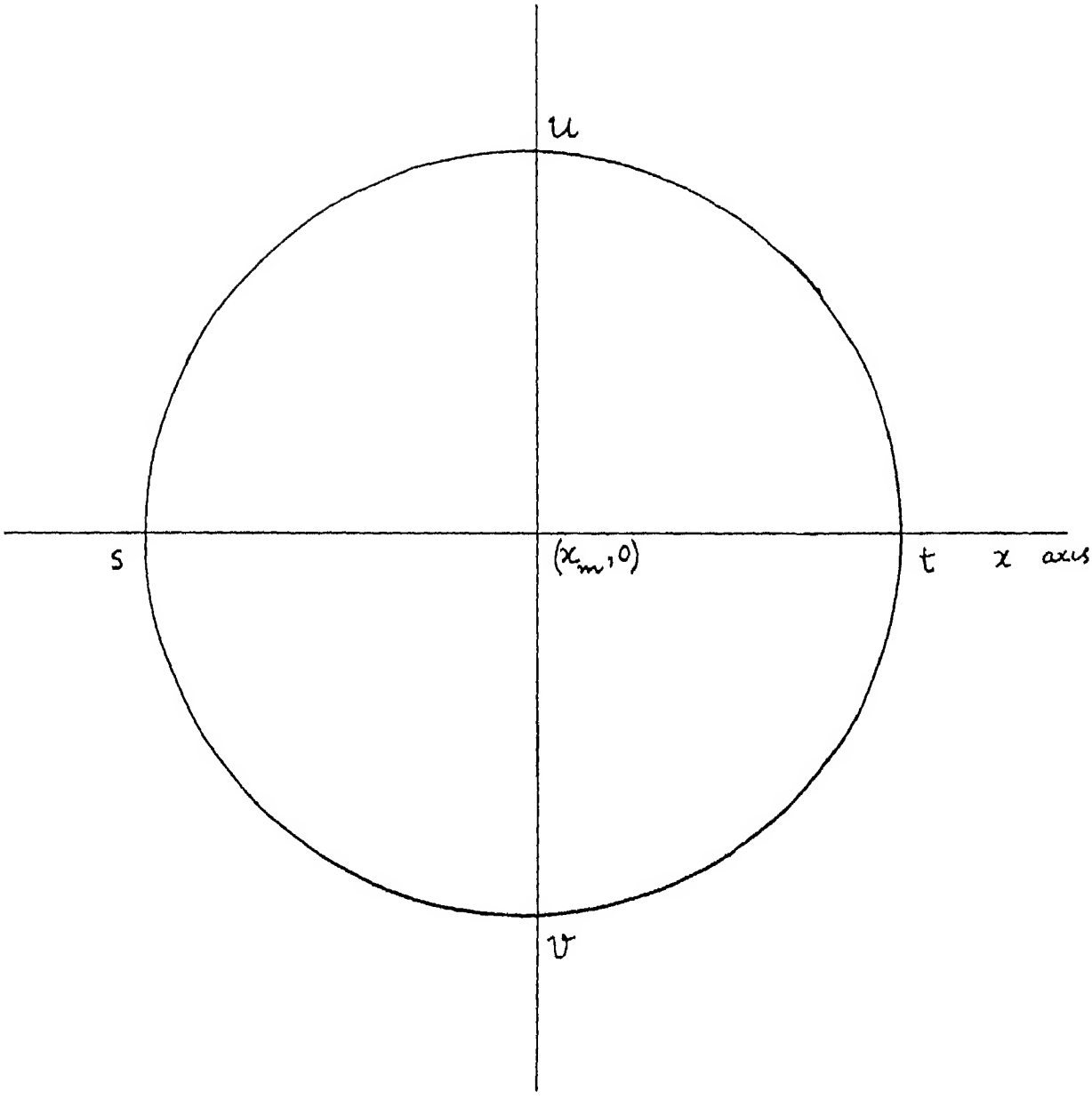


Fig 2 4 1

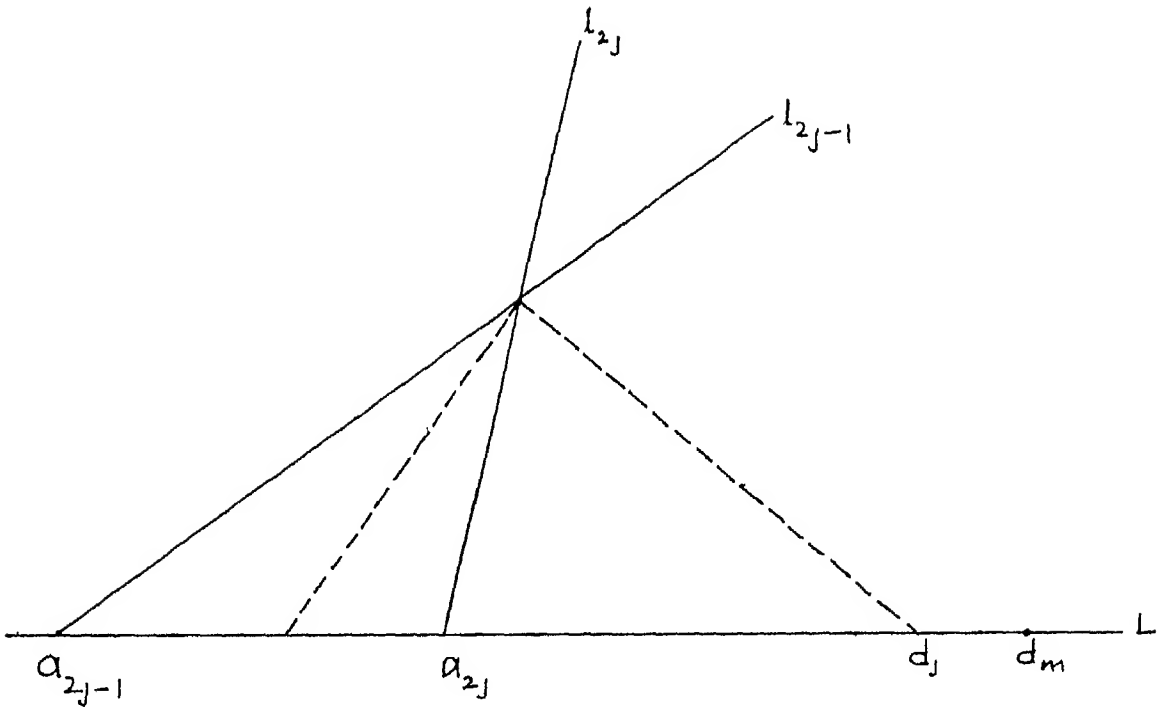


Fig 2 4 2 If $x^* > d_m$ discard l_{2j-1}

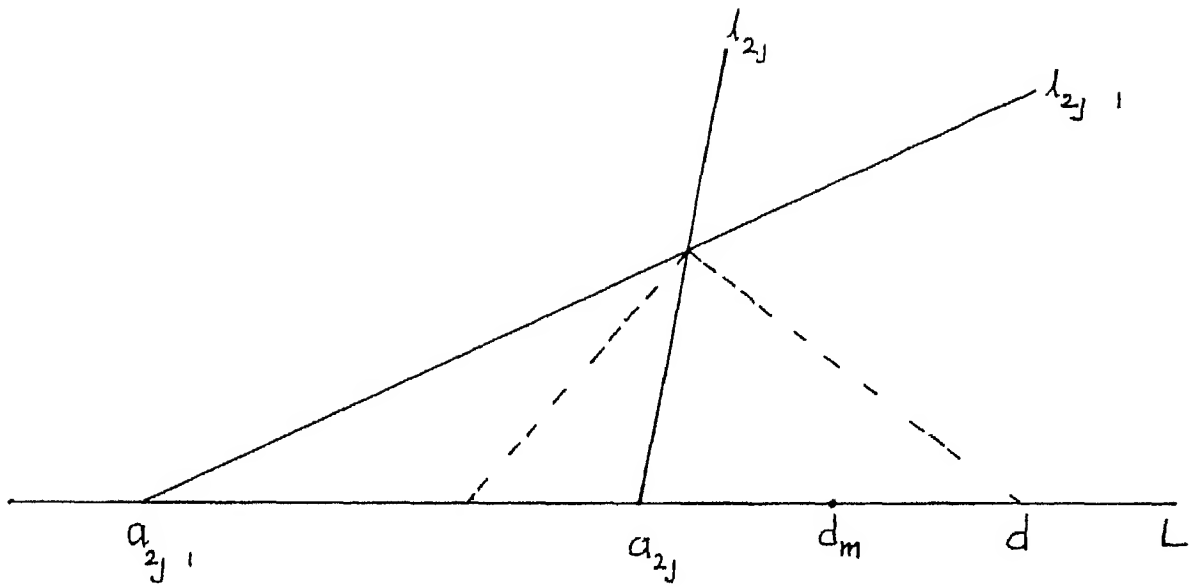


Fig 2 4 3 If $x^* < d_m$ discard l_{2j}

hese steps

Steps 3 5 and 11 tale $O(n^e)$ time by the analysis of *par-sel*

Step 8 tales $O(n^e)$ time This can be done by the method of arallel prefi computation

Step 1_ tales $O(n^e)$time by the method giver earlier

Altogether n/8 of the lines are discarded from consideration Therefore the time for this step is t(7n/8)

$$t(r) = t(7n/8) + c_1 n^e$$
$$- O(n^e)$$

Unconstrained center case

If (* Ø) is the center of the constrained problem using all the processors in parallel find the set

$$I \quad \{ i \quad |a_i * + c| / (a_i^2 + b_i^2) = g(>* \quad Ø) \}$$

If I is a singleton et then $y^* = a_1$ and y^c ha the sign of b_1 If I has e actly two elements then y^c has the sign of the y coordinate of the intersection of the corresponding lines Else if the lines corresponding to the indices in I form a bounded ccive set enclo ing (* Ø) then this is the required center cllerwisc y^c has the sign of the y-coordinate of the intersection point of the two lines such that to find the true center we need to move towards the intersection point of these lines along the intersection point of these lines along the bisector passing through (y* Ø)

The following procedure determines the center in the case of the unconstrained center problem

ar-uncons-line(S)

1) If $|S|$ is a constant use the sequential algorithm Else subdivide the input set of lines into subsequences of size n^{e} and assign a processor to each sequence

(2) For $i = 1$ to n^{1-e} do in parallel

P_i finds the angles each of the lines in its associated set makes with the x axis

end for

(3) Divide the lines into two sets L_1 L_2 with the lines in them making angles α β respectively with the x axis where $0 \leq \alpha \leq \pi/2$ and $\pi/2 \leq \beta \leq \pi$

(4) Find the lines l_m^1 l_m^2 the lines of median slope among the set L_1 and L_2 respectively

(5) Rotate the x axis so that l_m^1 is the new x-axis (Repeat this and the following steps with l_m^2 too)

(6) Use all the processors to pair up the lines $(l_i$ $l_j)$ so that one has a -ve and the other has a positive slope Find $(x_{ij}$ $y_{ij})$ the intersection point of these lines

(7) Find the median x_m of x_{ij}s

(8) Test on which side of the line $x = x_m$ the center lies using the method outlined above

(9) Assume the center lies to the left of the line (The other cases can be dealt similarly)

Find y_m the median of y_{ij} s which lie to the right of the line $x = x_m$

10) Determine the side of $y = y_m$ the true center lies Assume w l o g that it lies below this line (The center lies in the quadrant LL (fig 2 4 4))

11) B = { }

For i = 1 to n^{1-e} do in parallel

P_i considers all pairs of lines (l_i, l_j) in its associated set whose intersection points lie in the quadrant UR and add them to B

end for

(12) Repeat the above process with the set of lines in B which intersect LL to localise the center in one of the four quadrants created by the median lines Assume that this quadrant is the lower left quadrant L L let U R be the upper right quadrant

(13) Use all the processors in parallel to discard one of the pairs of lines in the given pairs whose bisectors donot intersect $IL \cap L L$ (fig 2 4 5)

(14) Let S be the set of remaining lines *par-uncons-l ne(S)*

2 4 3 Analysis

Step 1 takes constant time

Steps 2 5 6 11 12 and 13 take $O(n^e)$ time

Step 3 takes $O(n^e)$ time (by the analysis of step 4 of *par-sel*)

Steps 8 and 10 take $O(n^e)$ time by the method given earlier

The number of lines remaining for consideration are

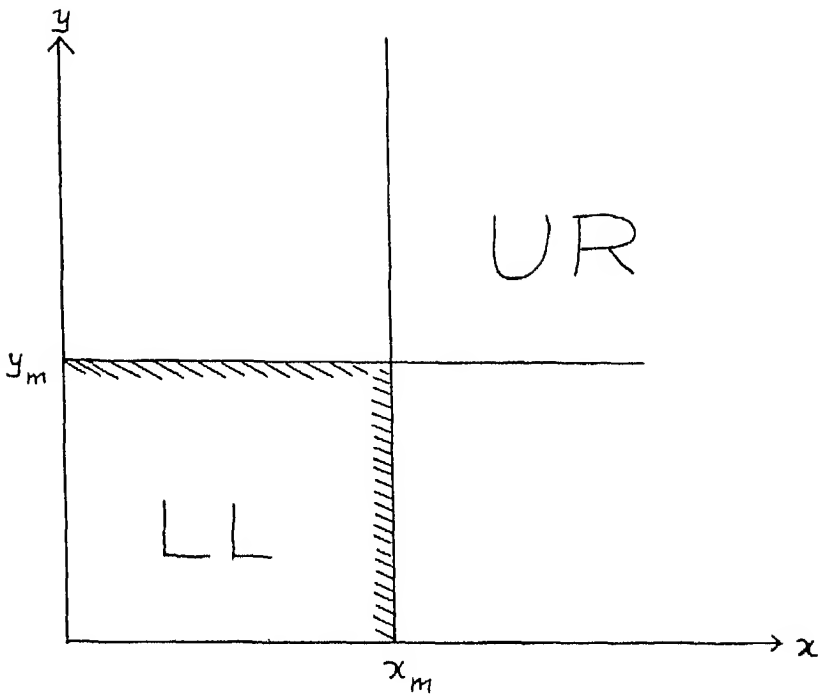


Fig 2 4 4 The center lies in the quadrant LL

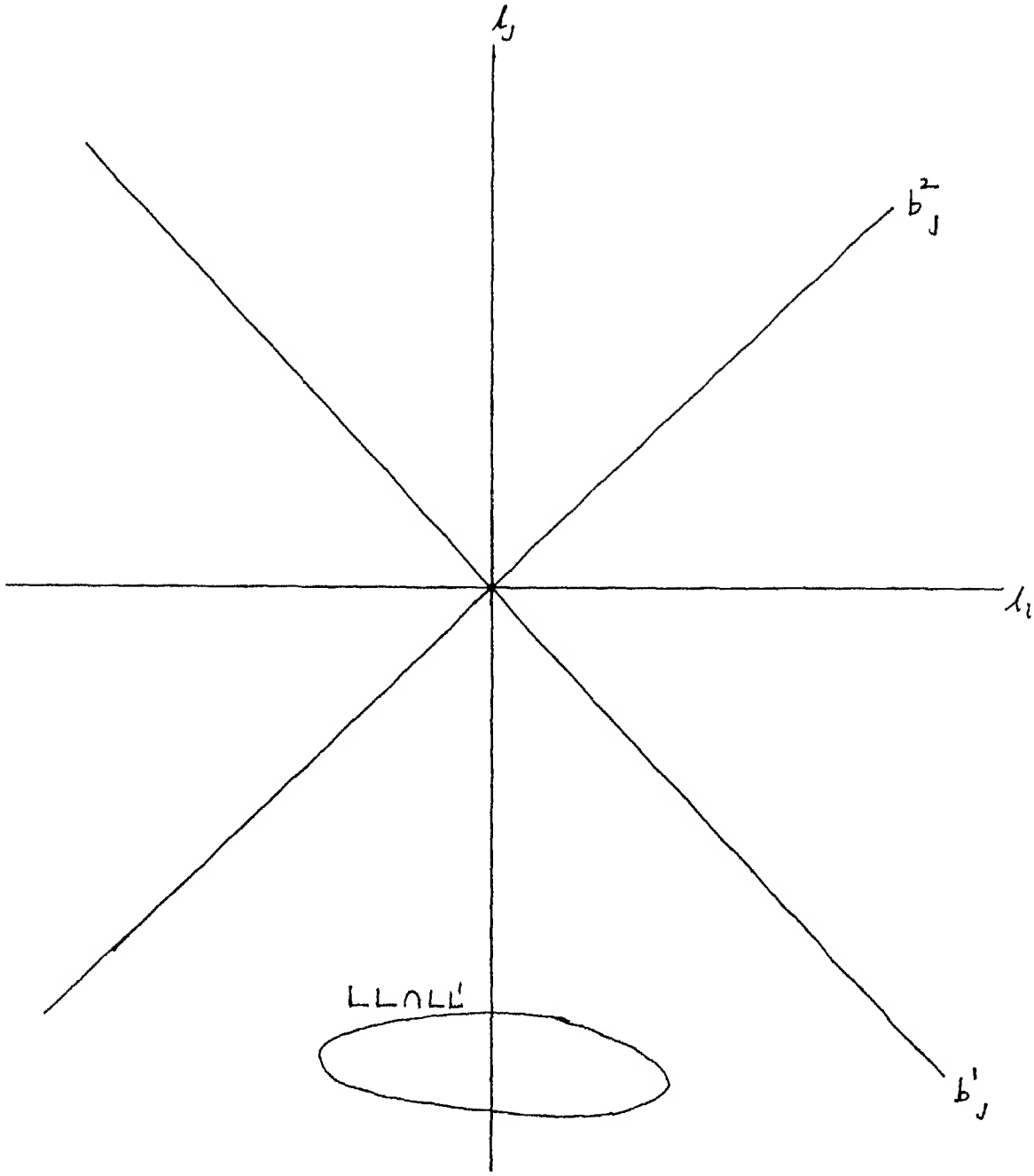


Fig 2 4 5

127n/128 Therefore step 14 takes t(127n/128)

$$t(n) = c_1 n^e + t(127n/128)$$
$$= O(n^e)$$

2.5 Minimum Spanning Circle for a set of line segments

The problem is to compute the smallest radius circle which intersects a given set of n line segments in the plane. This is solved by reducing the problem to that of computing the smallest radius circle for a set of lines and points and use a prune and search technique [JM90]. The following subproblems are solved

1. Given a set of n lines and points in the plane compute the smallest radius circle intersecting the e
2. Given a set of n rays in the plane show how these can be converted to a set of n points and lines

Then each line segment can be thought of as a degenerate intersection of two rays (fig 2.5.1) and hence we have subproblems 1 and 2.

We define

$$g_1(x, y) = \max_{(1 \le i \le n_1)} \{d_1(l_i(x, y)) \mid l_i \text{ in } L\}$$

$$g_2(x, y) = \max_{(1 \le i \le n_2)} \{d_2(p_i(x, y)) \mid p_i \text{ in } P\}$$

$$g(x, y) = \max(g_1(x, y), g_2(x, y))$$

where L is a set of lines, P is a set of points, $d_1(l(x, y))$ is the distance of the point (x, y) from l and $d_2(p(x, y))$ is the distance between points p and (x, y)

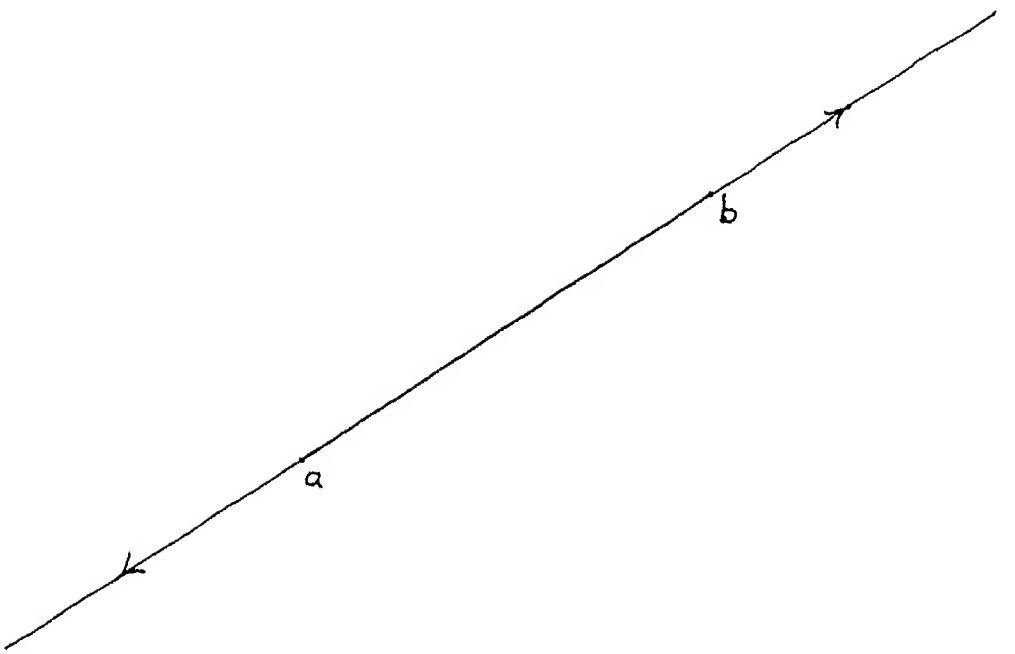


Fig 2 5 1 Segment $\overline{ab}$ is the intersection of the rays with tails at a and b

Solving subproblem 1

This problem is solved in similar lines as in the case of the problem with points and lines dealt with in the earlier sections We pair up the points of P arbitrarily and compute their bisectors Let $\rangle_{\pi}$ be the median of the intersection of the bisectors with the a is We compute g($_{\pi}$ ß) the radius of the smallest circle centered at (γ_{π} ß) We determine the side of ($\rangle_{\pi}$ ß) on which the true center lies depending on the points of contact of the lines which touch the circle and the points which lie on the circle Then we can throw away a fraction of the lines and points as outlined in the previous sections As given in the earlier sections we can determine which side of the given line the true certer lies The unconstrained problem is solved in much the same way Each pruning step consists of throwing away about 1/16th of the points and 1/64th of the lines Since all this can be done in $O(n^e)$ time using $O(n^{1-e})$ processors by the results of the previous sections we can say that subproblem 1 can be solved in $O(n^e)$ time using $O(n^{1-e})$ processors

Solving subproblem 2

Consider the constrained case first For each ray r_1 consider the line l_1 normal to it and passing through its tail (fig 2 = 2) We compute the median of the points of intersection of these normals with the \ a is and locate on which side of this median the true center lies Suppose the true center lies to the right of this median Then for each of the normals which

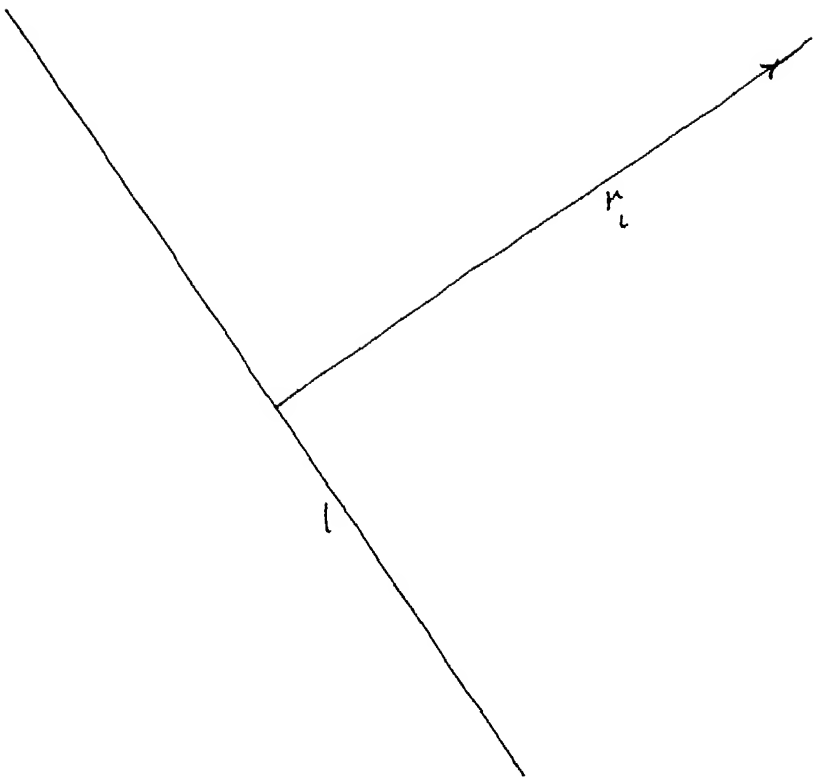


Fig 2 5 2 l_1 is a normal through the tail of ray r_1

nterrects the › a is to the left of this rredian point we replace he corresprnding ray by a line or a point If the ray and the edian point lie in the sarre half space of the two half spaces eterrined by the norrral then the ray is replaced by its line of upport else by its tail point This can be done in $O(n^e)$ tirre isirg $O(n^{1-e})$ processors by allowing each of these processors to work on subsequences of rays (each of si e n^e)

After doing this do the pruning step as indicated in the solution to the subproblerr 1 Iterate until a constant number of lines and points rerrain

Tc solve the unconstrained case we consider the norrals of the rays Rotate the › a/is to be parallel to the direction of the redian slope The rredian slope is the line whose angle with the › a is is the rredian of the angles of the lines with the a›is This rotation is to be done twice The first tirre we consider all the lines which rake an angle α with the › ayis where $\Pi \leq \alpha \leq \Pi/2$ The second tirre we consider all the lines which rake an angle β with the \ a is where $\Pi/2 \leq \beta \leq \Pi$ Pair up the norrals with the ve and +ve slopes arbitrarily and consider the set of intersection› ($_{1J}$ y_{1J}) Let $›_{\pi}$ y_{π} be the rredians of $›_{1J}$ s and y_{1J} s respectively Locate the center w r t $› = ›_{\pi}$ and $y = y_{m}$ Let the center lie in the lower left quadrant LL (fig 2 5 3)

Of the set of norral pairs which have their intersections in the upper right(UR) quadrant one of each does not intersect quadrant LL We have a replacerrent strategy for the corresponding

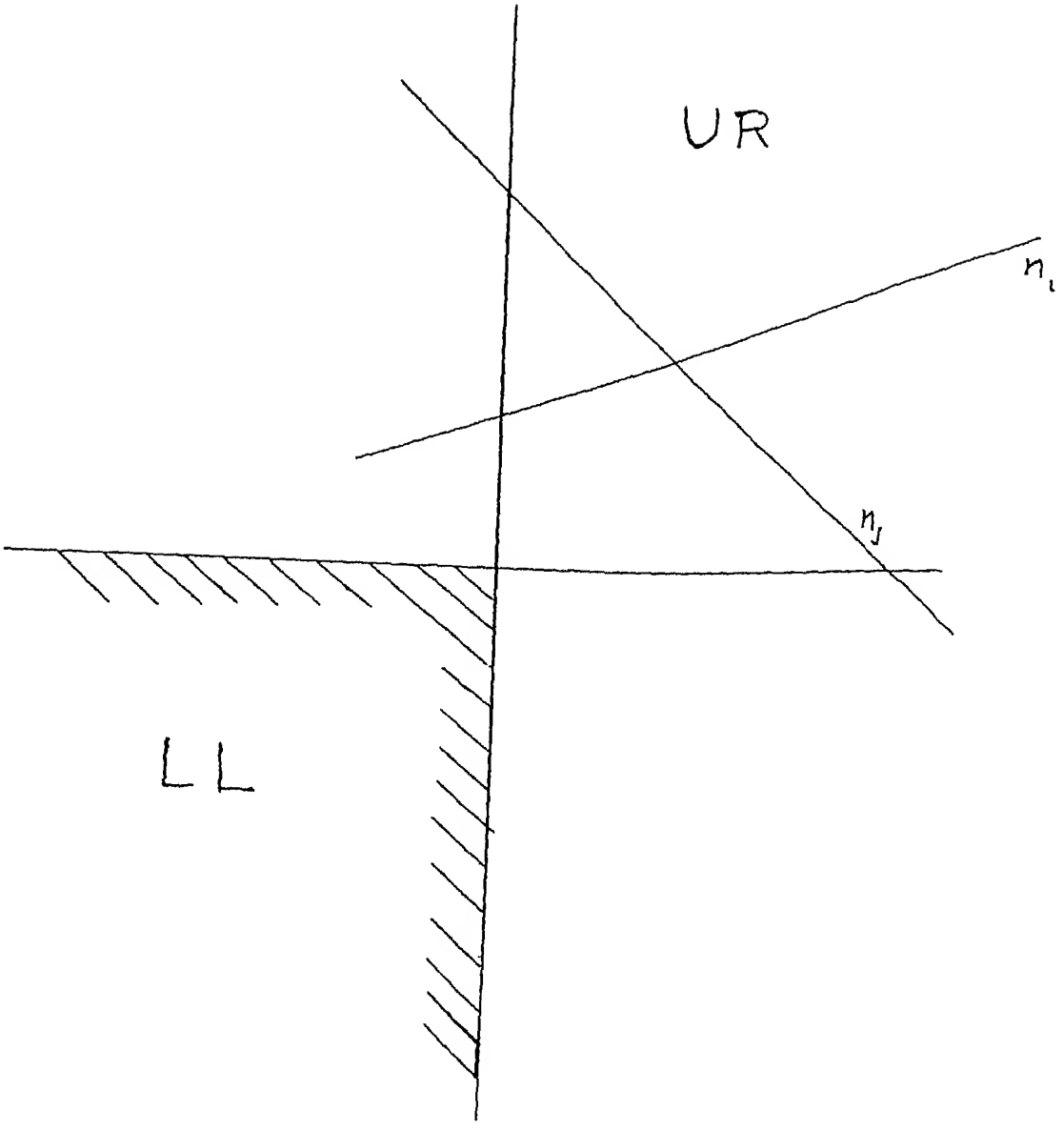


Fig 2 5 3 Normal n_j does not intersect LL

rays of these normals If the ray and LL lie in opposite half spaces determined by the normal then replace the ray by its tail point else replace it by its supporting line (fig 5 4) Thus in a single iteration we replace 1/8 th of the rays by lines or points Note that all this can be done in $O(n^e)$ time using $O(n^{1-e})$ processors by allowing each processor to work on a subsequence of size n^e

After replacing a fraction of the rays we do pruning (subproblem 1) on the set of lines and points generated so far Iterating the e processes we can solve the problem in $O(n^e)$ time using $O(n^{1-e})$ processors

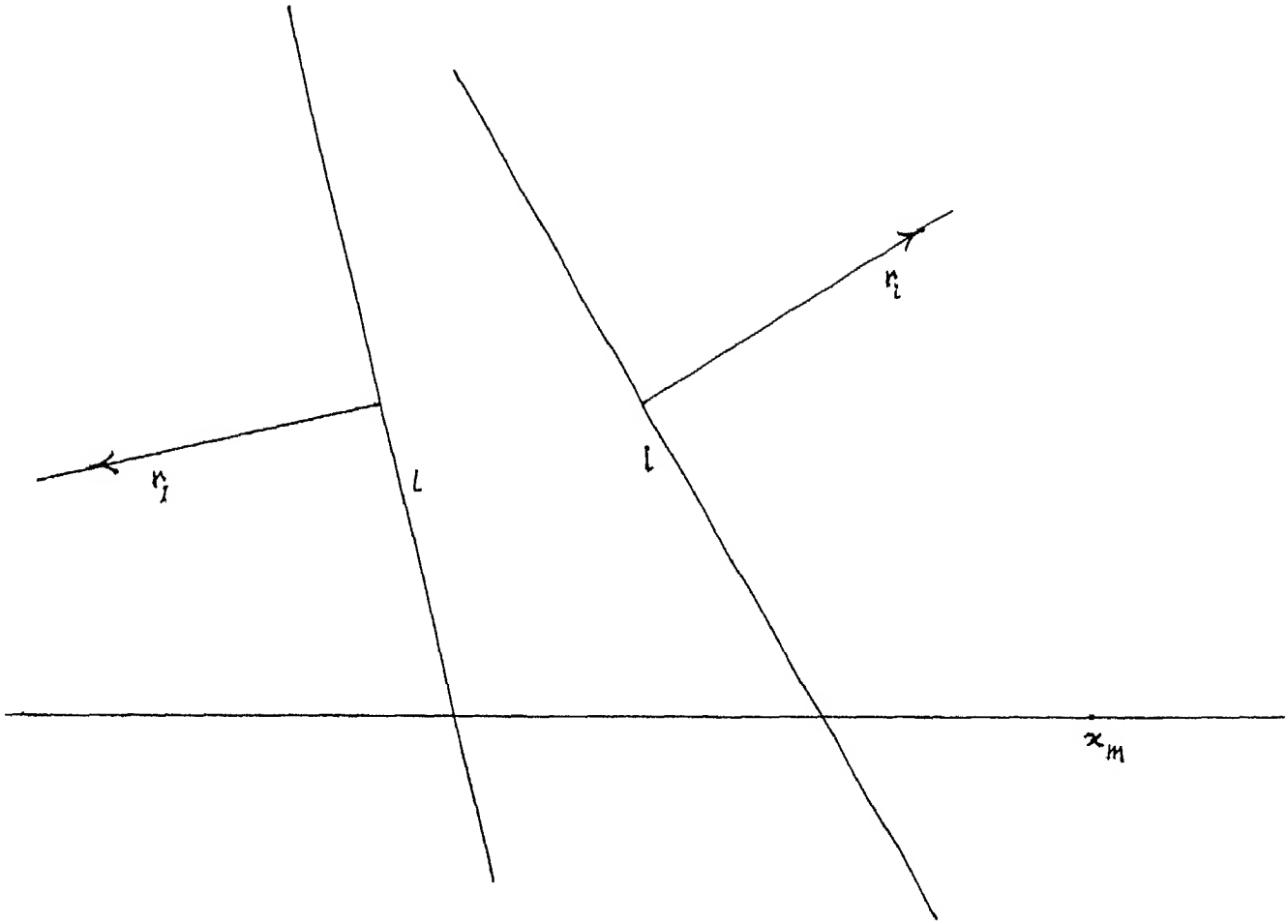


Fig 2 5 4 Ray r_1 is replaced by the supporting line of r_i Ray r_1 is replaced by its tail point

CHAPTER 3

COMPUTING MINIMAL LENGTH LINE SEGMENT OF EXTERNAL VISIBILITY

3 1 Introduction

Though the notion of weak visibility has been well studied the notion of external weak visibility hasn t been given much attention It is shown in [BT89] that given a convex polygon P the minimal length line segment from which P is weakly externally visible can be found in O(n) time The algorithm is based on the solution to a geometric minimisation problem

In this chapter we give a parallel algorithm to compute the shortest line segment from which a given convex polygon is weakly external visible in $O(n^e \log n)$ time using $O(n^{1-e})$ CREW PRAM processors In section 3 2 certain definitions and lemmas are given In section 3 3 the parallel algorithm is proposed which is analysed in section 3 4

3 2 Preliminaries

For any integer $n \geq 3$ we define a ***polygon*** in the Euclidean plane E^2 a the figure $P = [p_1 \ p_2 \ \ldots \ p_n]$ formed by n points p_1 p_2 $\ldots$ p_n in E^2 and n line segments $[p_i \ p_{i+1}]$ $i = 1, 2, \ldots, n-1$ and $[p_n \ p_1]$ The points p_i are called the ***vertices*** of the polygon and the line segments are termed its ***edges***

We assume that the vertices of P are in ***general position*** i e no three vertices are collinear and that the polygon is in ***standard form*** i e the vertices appear in counterclockwise order

A simple polygon P is said to be *weakly externally visible* from a line segment L if L is outside P and for every point x on the boundary of P there is a point y on L such that the interior of the line segment [x y] doesn't intersect P

HL(x y) denotes the closed *half-plane* to the left of the directed line determined by the two ordered points The corresponding closed half plane to the right of the directed line determined by two ordered points is denoted by HR(x y)

The *edge* $[p_i\ p_{i-1}]$ of P is denoted by e_i If the half plane contains the interior of P it will be referred to as an interior half plane We denote the infinite half ray starting at point x and traversing a second point y by *ray(x y)*

The interior cone of support at p_i denoted by *i-cone(p_i)* is the wedge determined by $HL(p_{i\ 1}\ p_i) \cap HL(p_i\ p_{i+1})$ (fig 3 1)

A *line of support* of a convex polygon P is a line that has at least one point in common with P but with the property that all of P lies on one side of the line

A pair of vertices of the polygon that admits parallel supporting lines will be called *antipodal*

In fig 3 2 p_4 and p_6 are not antipodal since parallel lines of support can't pass through p_4 and p_6 simultaneously p_5 and p_1 form an antipodal pair

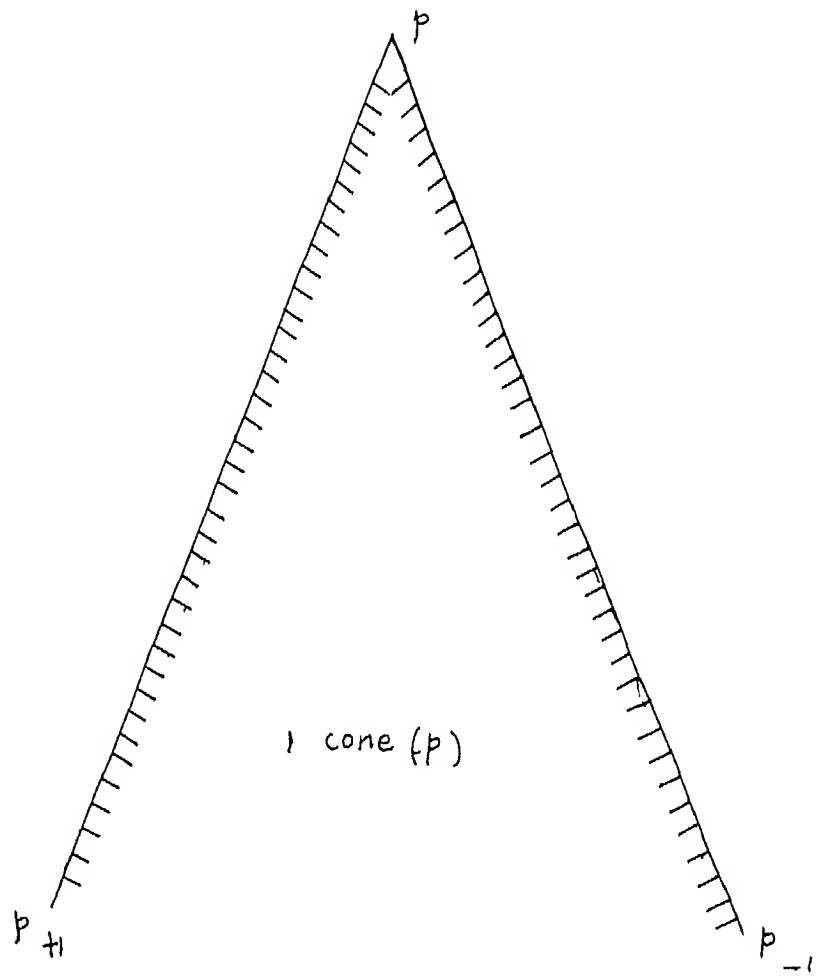


Fig 3 1 Illustrating the internal cone of support of vertex p_1 of a convex polygon P

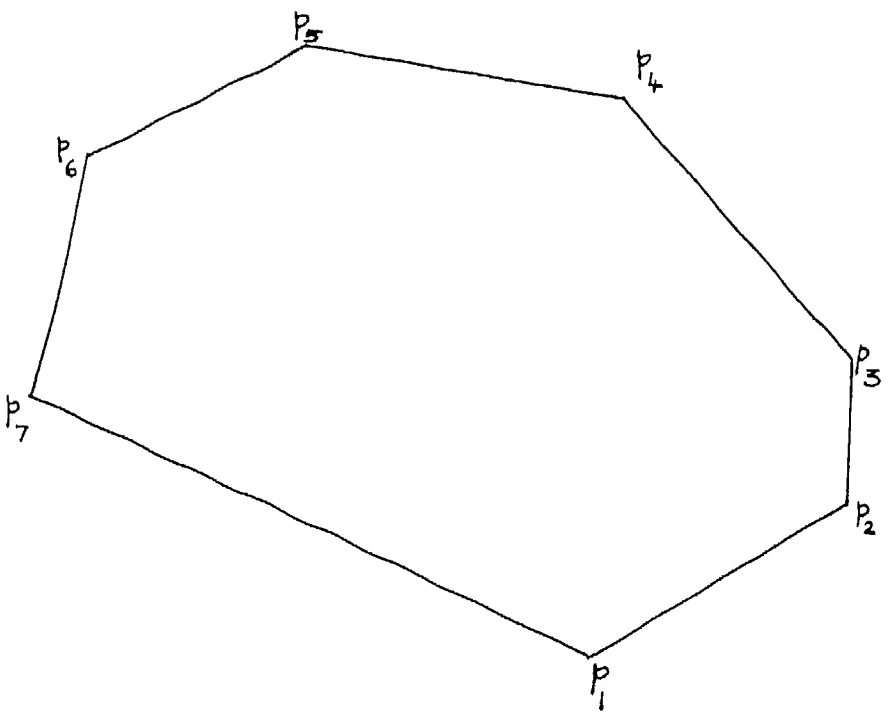


Fig 3 2 p_4 and p_6 are not antipodal while p_1 and p_5 are an antipodal pair

Let $L^{+}(P)$ be a line segment from which the polygon P is weakly externally visible Let $L^{*}(P)$ be the shortest of all such $L^{+}(P)$

Lemma 3 2 1 $L^{*}(P)$ is tangent to P

Lemma 3 2 2 $L^{*}(P)$ has one of its endpoints on one of the bounding rays of i cone(p_i) and the other on the other bounding ray for some value of i ($1 \le i \le n$)

Lemma 3 2 3 : P is weakly externally visible from L[a b] if and only if ∃ tangent rays of support to P from a and b ray(a P) and ray(b P) such that the following three conditions hold

(i) ray(a P) and ray(b P) intersect at some point

(ii) v is a vertex of P and

(iii) P is contained in Δabv

Lemma 3 2 4 A line segment L = [a b] lying in i cone(p_i) tangent to P from which P is weakly externally visible must be tangent to a vertex p_j such that p_j is an antipodal vertex of p_i

Proof If L does not intersect ray (p_i p_{i-1}) then edge [p_{i-1} p_i] of P is not visible from L Similarly if L does not intersect the ray(p_i p_{i+1}) then edge [p_i p_{i+1}] of P is not visible from L Therefore L must have one end point on the ray (p_i p_{i-1}) and the other on the ray(p_i p_{i+1}) Now assume that p_j is not an antipodal vertex of p_i Let $p_{k(i-1\ i)}$ denote the vertex of P determined by a tangent line parallel to the ray(p_i p_{i-1}) and antipodal to p_{i-1} and p_i Similarly let $p_{k(i\ i+1)}$ denote the vertex of P determined by a tangent line parallel to ray(p_i p_{i+1}) and antipodal to p_i and

P_{i+1} Clearly as we rotate a line of support in a counter clockwise manner starting at $P_{h(i\ 1\ i)}$ and ending at $P_{h(i\ i+1)}$ we visit the polygonal chain C $[P_{h(i\ 1\ i)}\ P_{h(i\ 1\ i)+1}$ $P_{h(i\ i+1)}\ 1\ P_{h(i\ i+1)}]$ which has the property that all its vertices are antipodal to P_i Furthermore if P_j is not antipodal to P_i then L must have an unoriented direction that lies in the wedge determined by the internal angle of P at the verte P_i This implies that one endpoint of L must lie in int(i cone(P_i)) which is a contradiction

Lemma 3 2 5 Let p be any point in the interior of the i cone of a verte P_i the length of the line segment [H K] such that H and K lie on the boundary rays of i cone(P_i) and passing through point p is a unimodal function and the minimum can be found in constant time [BT89]

Lemma 3 2 6 The total number of antipodal pairs is N The set of all antipodal pairs can be found in linear time [SM75]

Proof The set of all antipodal pairs can be determined as follow Tran late the edges of the polygon to the origin treating them as vectors In this mapping edges go to vectors and vertices go to sectors (fig 3 3) In order to find the antipodal pair corresponding to some direction determined by a line L translate the line so that it passes through the origin of the vector diagram The sectors through which it passes through indicate the points of the antipodal pair Determining the sectors which the line passes through can be done in O(log n) time The set of all

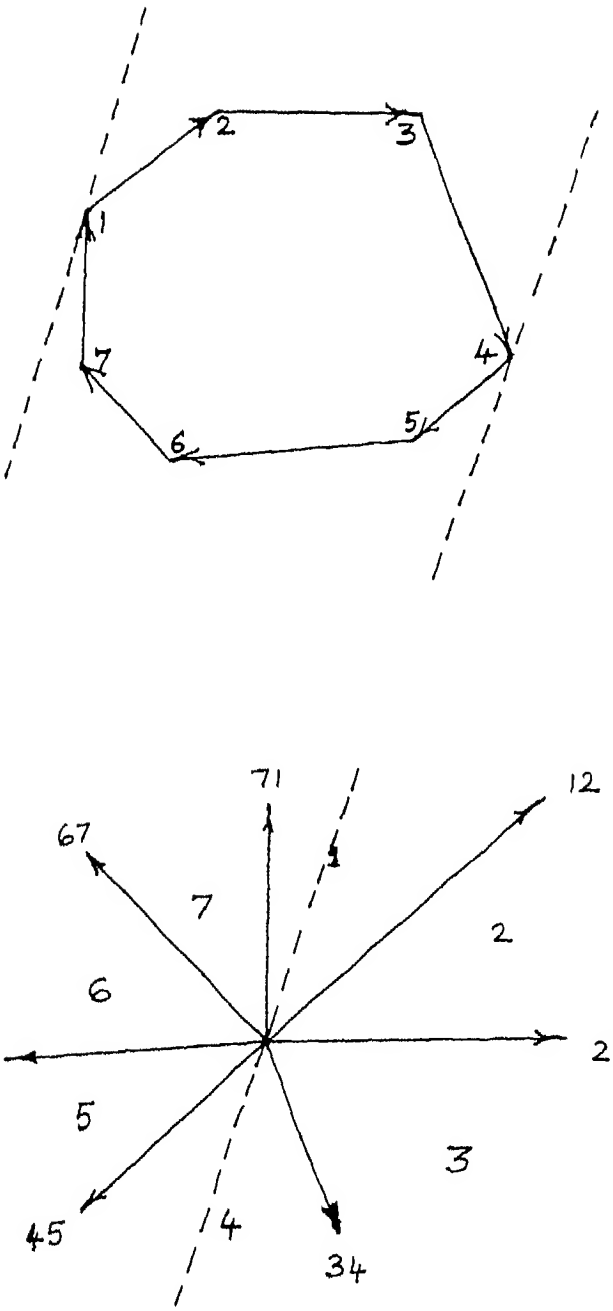


Fig 3 3 Determining antipodal pairs

antipodal pairs can be found out in O(n) time by scanning sequentially around the vector diagram

3 3 The Algorithm

Lemmas 3 2 1 3 2 2 3 2 3 and 3 2 4 imply that any $L^{+}(P)$ (a line from which a polygon is weakly external visible) must be tangent to a vertex p_j and have its end points on the bounding ray of $cone(p_1)$ such that p_1 and p_j form an antipodal pair

In order to find $L^{+}(P)$ through a vertex p_1 we need to know the bounding rays on which its endpoints lie in order to apply lemma 3 These bounding rays are determined by $cone(p_j)$ where p_j and p_1 are antipodal pairs Our algorithm finds out all such possible $L^{+}(P)$ in parallel and finds the one of minimum length among those

The following procedures run on a shared memory SIMD machine with $n^{1-\epsilon}$ processors P_1 P_2 $P_{n^{1-\epsilon}}$ where $0 < \epsilon < 1$

par-short-vis-line computes the shortest line segment from which a given convex polygon is weakly external visible using $n^{1-\epsilon}$ CREW PRAM processors The input to the procedure is the polygon P (a set of vertices V and a set of edges E) The output is the shortest line segment from which the polygon is weakly external visible

par-short-vis-line(V E)

(1) Subdivide the sequence of edges E into $n^{1-\epsilon}$ subsequences each of size n^{ϵ} and assign a subsequence to each processor

( ) For i = 1 to n^{1-e} do in parallel

P_i translates each of the edges in its associated subsequence to the origin treating them as vectors

end for

(3) For i = 1 to n^{1-e} do in parallel

P_i determines the antipodal pairs corresponding to the direction of each of the edges in its associated subsequence by determining the sectors through which the lines in those directions pass through

end for

(4) For k := 1 to n^{1-e} do in parallel

For each antipodal pair $(p_i \; p_j)$ associated to P_k P_k find the shortest line segment through p_i with its end points on the bounding rays of i cone(p_j) If this line is not a line of support to P (i e it cuts the interior of P) P_k takes the shorter of the line segments through $(p_{i-1} \; p_i)$ and $(p_i \; p_{i+1})$ with the end points on the bounding lines of i cone(p_j) (fig 3 4) P_k similarly find the line segment through p_j P_k then finds the shortest of the line segments that it has found using the antipodal pairs allocated to it and writes it in S[k] the k^{th} position of an array S in shared memory

end for

end for

(5) Call *par-short(S)* to find the shortest line segment

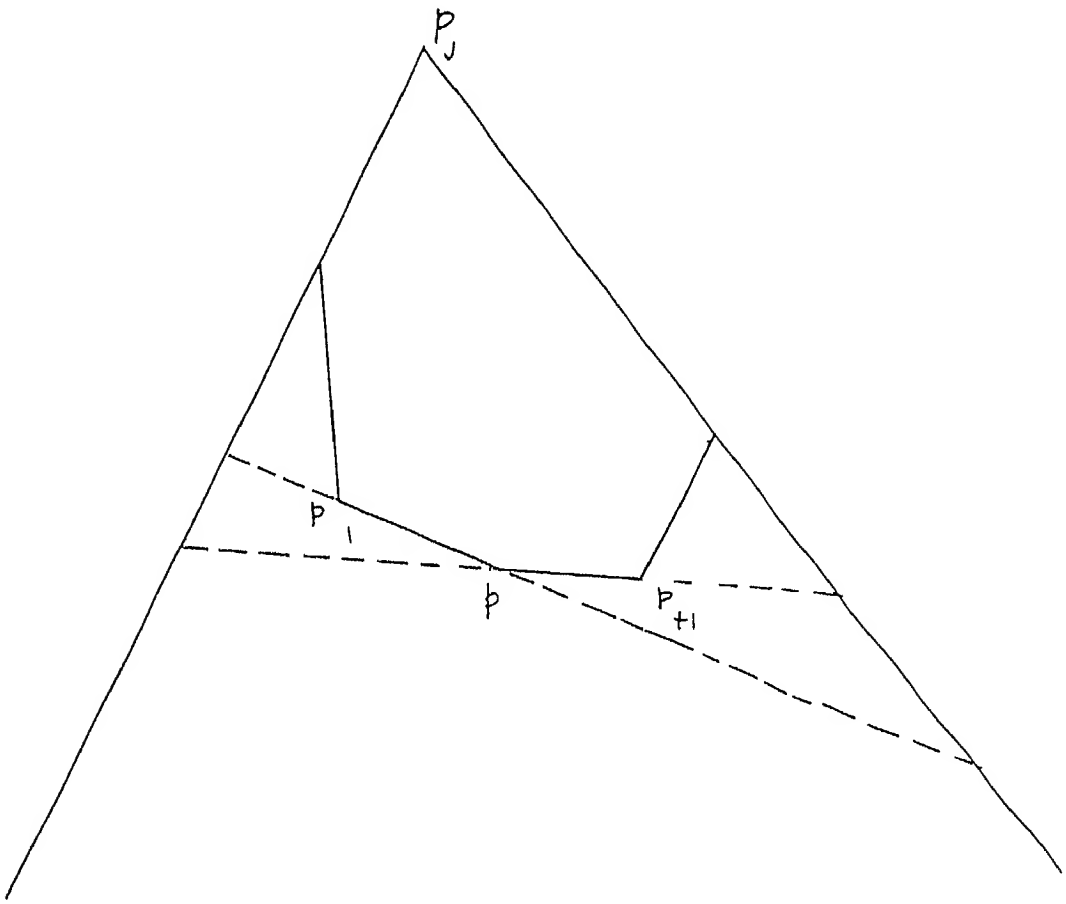


Fig 3 4

The following procedure finds the shortest of the given set of line The input is a set of line segments along with their lengths

The output is the shortest line segment

par-short(S)

1 If $|S|$ 3 then use one processor to return the shortest line

else

Subdivide S into $|S|^{1-e}$ subsequences each of which has $|S|^{e}$ elements and assign a subsequence to each processor

2 For i = 1 to $|S|^{1-e}$ do in parallel

Processor P_i finds the shortest s_i of its associated subsequence using the sequential algorithm for finding the shortest

P_i writes s_i in M(i) the i^{th} position of an array M in the shared memory

3 *par-short(M)*

3 4 Analysis

Analysis of *par-short*

Let t(n) be the running time of *par-short* for an input of size n

Step 1 takes constant time

The sequential algorithm to select the shortest is of linear time Therefore step 2 takes $c_1 n^{e}$ time units where c_1 is a constant

Since $|M| = n^{1-e}$ step 3 requires $t(n^{1-e})$ time

From the above we have

$t(n) = c_1 n^e + t(n^{1-e})$ whose solution is $t(n) = O(n^e)$

Analysis of *par short-vis-line*

Step 1 needs constant time

Step 2 takes $c_1 n^e$ time units where c_1 is a constant because the translation of a single edge takes constant time and there are n^e edges associated with each processor

In step 3 the antipodal pairs are found by initially determining the antipodal pair corresponding to the direction of the beginning edge in the associated set of edges of a processor The antipodal pairs corresponding to the directions of the rest of the edges is found by a linear scan To find the antipodal pair corresponding to the first direction it takes O(log n) time according to lemma 3 2 6 To find the rest of the antipodal pairs it takes $O(n^e)$ time Hence the time for this step is dominated by n^e Therefore the time for this step is $c_2 n^e$ where c_2 is a constant

In step 4 each processor takes constant time to find the line segment corresponding to one antipodal pair according to lemma 3 2 5 Therefore this step requires $c_3 n^e$ time units where c_3 is a constant

Step 5 needs $c_4 n^e$ time units where c_4 is a constant according to the analysis done above

Therefore $t(n) = O(n^e)$

CHAPTER 4

RECONSTRUCTION OF AN ORTHOGONAL POLYGON FROM ITS VISIBILITY GRAPH

4 1 Introduction

The recognition problem for visibility graphs is given a graph to determine whether it is the visibility graph of a simple polygon The complexity of this problem is unknown In [OJ87] a linear time algorithm has been given for the reconstruction of an orthogonal polygon given the circular embedding of its visibility graph In [E(90] a linear time algorithm has been given for the recognition of the visibility graph of a spiral polygon

In the present chapter a parallel algorithm is proposed for the problem of constructing an orthogonal polygon given its visibility graph The algorithm runs on O(p) CREW PRAM processors in O((log p)n/p) time In section 4 2 preliminaries are given A sequential algorithm is briefly reviewed in section 4 3 In section 4 4 a parallel algorithm is proposed which is analysed in section 4 5

4 2 Preliminaries:

In this section certain definitions and notations are given

A *polygon* is a finite figure in the plane that is bounded by a finite number of straight line segments

A singly connected polygon is bounded by n points V_1

v_2 ... v_n called *vertices*) and n line segments $[v_1\ v_2]$ $[v_2\ v_3]$ ... $[v_{n-1}\ v_n]$ and $[v_n\ v_1]$ (called *edges*)

An *orthogonal polygon* is a polygon whose edges are all aligned with a pair of orthogonal co ordinate a es which we take to be the horizontal and the vertical without loss of generality Thus the edges alternate between hori ontal and vertical and always meet orthogonally with internal angles of either 90 or 270 (Fig 4 1 is an orthogoral polygon)

An orthogonal polygon is said to be in *general position* if no two verti es can be connected by an internal hori ontal or vertical line segment that doesn t intersect P s boundary Throughout the discussion here we use polygon to mean orthogonal polygon in general position

Two vertices v_i and v_j of a polygon P are *visible* if the clo ed line segment between them doesn t intersect the e terior of P Note that if the line segment touches the boundary of P the vertices are still considered visible

Two edges e_i and e_j are said to be *horizontally (vertically) visible* iff they can see one another along a vertical (hori ontal) line that is iff there exists a vertical (hori ontal) line segment interior to P with end points on e_i and e_j and which doesn t otherwise intersect the boundary of P (In fig 4 1 edges 0 and 4 are visible while edges 1 and 2 are invisible)

An edge is said to be a *bottom(top)* edge if the interior of the polygon lies *above(below)* it Similarly it is said to be a

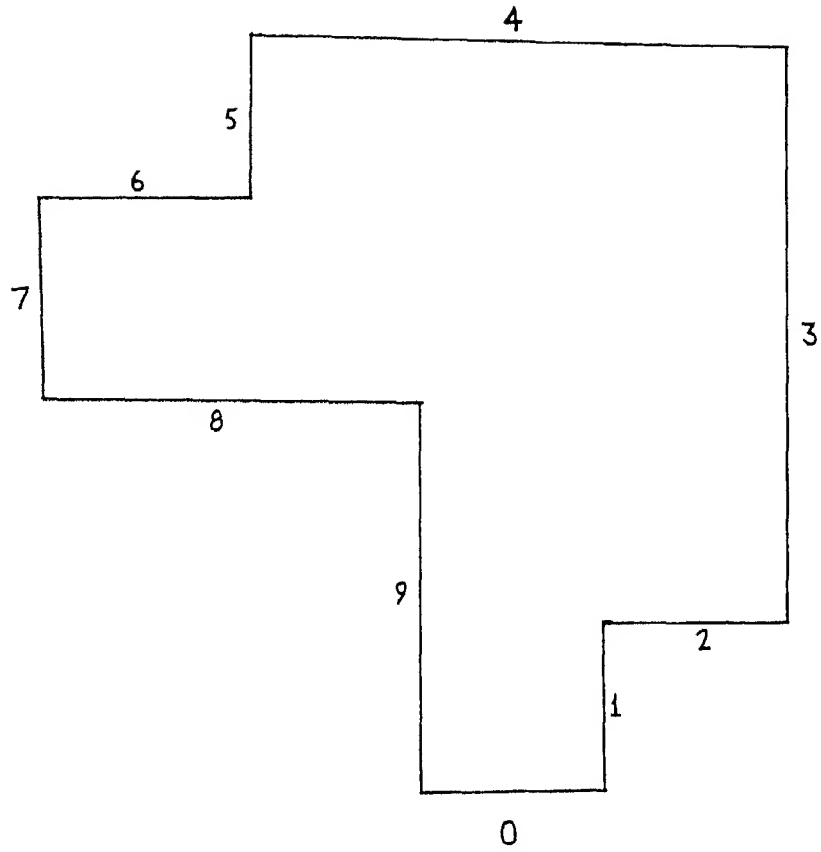


Fig 4 1 n orthogonal polygon

left(right) edge if the interior of the polygon lies to its *left(right)* (In fig 4 1 edges 0 and 4 are bottom and top edges r pectively)

A vertical edge $[(x_1 \ y_1) \ (x_1 \ y_2)]$ is said to be *going up* if $y_1 \ y_2$ else it is said to be *going down* A hori ontal edge $[(\ _1 \ y_1) \ (x_2 \ y_1)]$ is said to be *going left* if $_1 > x_2$ else it is said to be *going right*

The nodes of a *visibility graph* correspond to geometric components such a vertices or edges and two nodes are conne ted by an arc of the graph if the components can see one another perhaps under some restricted form of visibility

The *orthogonal edge visibility graph* G for an orthogonal polygon P is defined as follows G contains a node for each edge of P and two nodes associated with hori ontal(vertical) edges e_1 and e_J are connected by an undirected arc in G iff they are hori ontally(vertically) visible In the discussion here the term visibility graph is used for the term orthogonal edge visibility graph

The visibility graph of an orthogonal polygon con sists of two disjoint trees one corresponding to the hori ontal visibility (called the *horizontal visibility tree* T_H) and the other corresponding to the *vertical visibility tree* T_V)

We say that a tree is *realisable* if there is a polygon with the tree as one of the two components of its visibility graph

We say that two trees can *mesh* if they are jointly realisable

by the e n polygon

A *labeling* of a tree of n nodes is a bijection between th nodes and the set of integers 0 1 n-1

A labeling of a tree is realisable if there is a polygon that realises the tree and such that the polygon edges may be numbered 0 1 n 1 in a counter clockwise traversal of the boundary to agree with the labeling

Define a *circle embedding* of a tree as a layout of the tree within a circle in the plane such that each arc is mapped to a chord of the circle and all nodes of the tree are mapped to lie on the circle (fig 4 2) The circle corresponds to the boundary of the polygon inflated to a circle and an embedding is topologically equivalent to a layout of the visibility trees within the polygon itself A labeling of a tree maps to a unique embedding

The portion of the polygon that realises the subtree from a node to its immediate descendants is a *histogram* or Manhattan skyline polygon This polygon is the orthogonal edge visibility polygon for edge e enclosing all the points visible to edge e

Two nodes of an embedded tree are said to be *2-adjacent* or *neighbours* if they are adjacent on the circle if the other tree is ignored Note that 2 adjacent nodes receive labels i and i+2(mod n) in the labeling

The *distance* between two 2 adjacent nodes is defined as the number of arcs in the path in the tree between them Let d_i be the

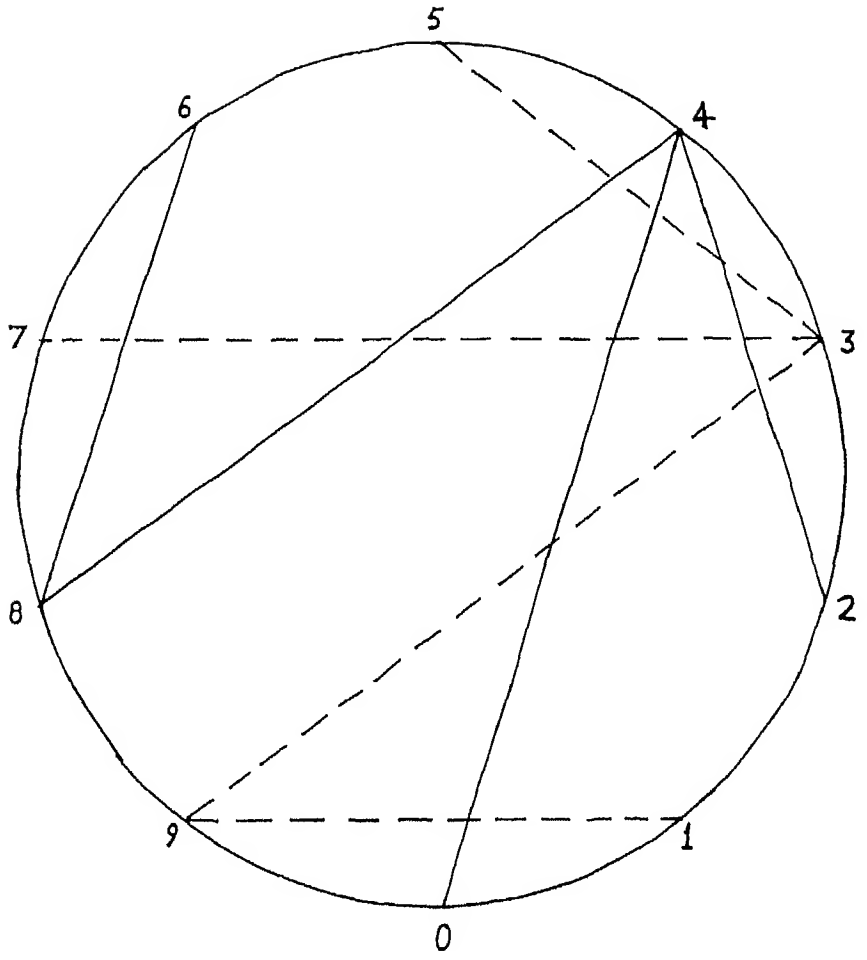


Fig 4 2 Circle embedding of the visibility graph

distance between node i and its counter clockwise neighbour

Lemma 4.2.1 In a realizable embedding the distance d_i between each pair of 2 adjacent nodes satisfies $d_i \leq 3$. Moreover the two angles in a realizing polygon between two adjacent nodes are determined by d_i as follows where c and r mean convex and reflex angles respectively

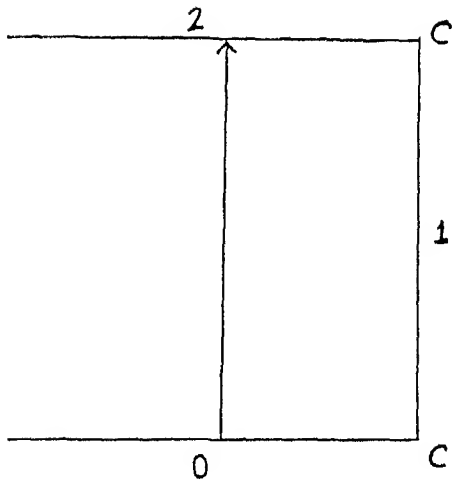
$$d_i = 1 \quad cc$$

$$d_i = 2 \quad rc \text{ or } cr$$

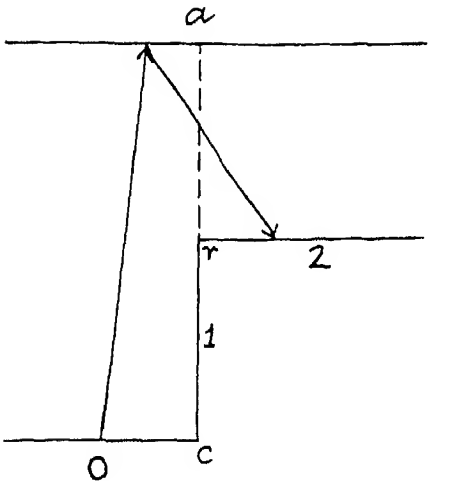
$$d_i = 3 \quad rr$$

Proof - Let an arbitrary horizontal edge of a polygon which we take to be a bottom edge without loss of generality be labeled 0 and label the remaining edges with increasing index counter-clockwise. The right end point of 0 is either a convex or a reflex vertex. If this end point is convex then distinguish two further cases depending on whether the upper end point of 1 is convex or reflex. The former case is illustrated in fig. 4.3(a) and justifies the claimed correspondence between $d_i = 1$ and cc. In the later case (fig. 4.3(b)) there must be an edge a above 1 leading to $d_i = 2$ and angles cr.

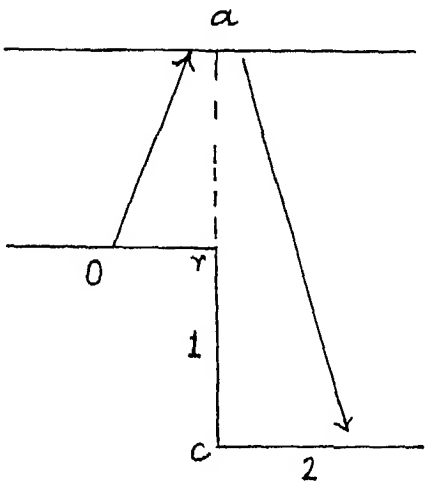
If the right end point of 0 is reflex we again have two cases depending on whether the lower end point of 1 is convex or reflex. In the former case (fig. 4.3(c)) there is again an edge a above 1 and $d_i = 2$ with angles rc. In the latter case (fig. 4.3(d)) there must be an edge a above 1 and an edge b below



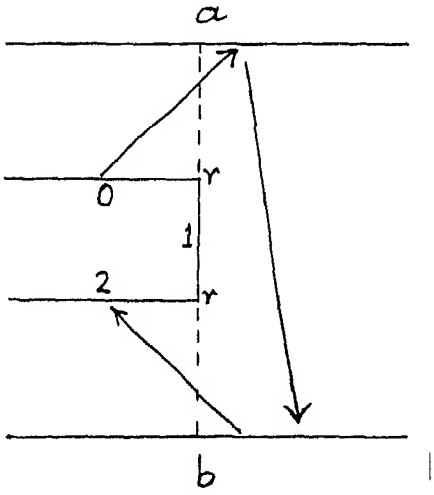
(a)



(b)

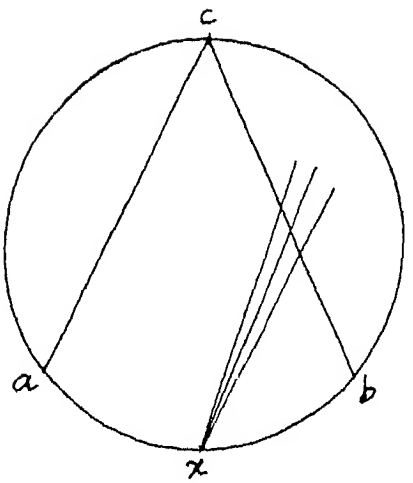


(c)

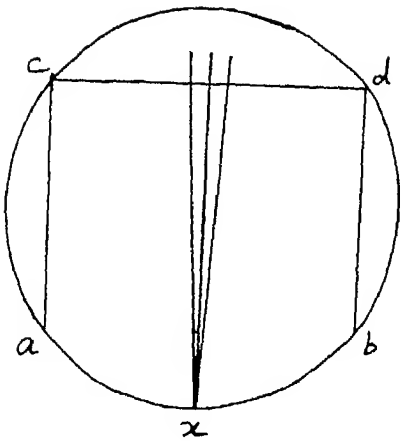


(d)

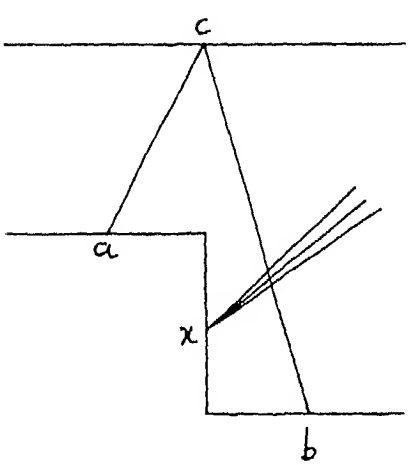
Fig 4 3 Determining angle sequences



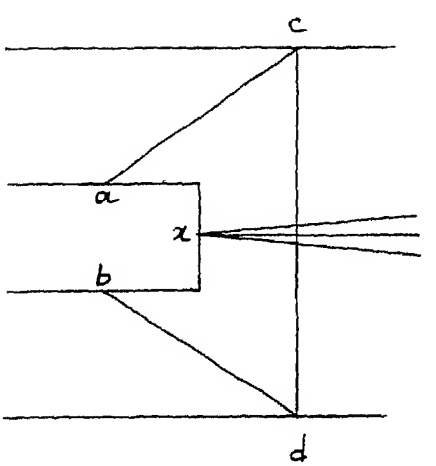
(a)



(b)



(c)



(d)

Fig 4 4 Illustrating projection constraints

This algorithm identifies which edges of the orthogonal polygon determine the vertical edges of the edge visibility polygon. Once these are identified we know their left to right ordering.

Let e be a bottom edge whose node has degree greater than one and let a and b be two edges visible from e with the edges occuring in counterclockwise order e a b and with no edges between a and b also visible to e. Then the vertical edge of e's edge visibility polygon between a and b is determined by either a+1(mod n) or b-1(mod n). Which case obtains can be decided by checking whether node a+1(mod n) is connected by an arc to any node in the range b counterclockwise to e. If node a+1(mod n) is connected to a node in the range b counterclockwise to e then a+1(mod n) is the vertical edge of e's edge visibility polygon else b-1(mod n) is the vertical edge. Continuing in this manner we can conclude that $e_1 < e_2 < \cdots < e_j$ where a < b means that the x coordinate of edge a is less than that of b and e_1 e_2 $\cdots$ e_j are the vertical edges of the edge visibility polygon of edge e where e is a bottom edge. If e is a top edge the ordering would have been e_1 e_2 $\cdots$ e_j. After all these partial orders have been determined they are merged. The position of an edge in this list is used as the x coordinate for the vertical edge. Similarly the y coordinate for the horizontal edge can be determined. Thus the polygon can be constructed.

4.4 The parallel algorithm

A straight forward parallelization of the above method would not yield an efficient algorithm. Instead we take up a different approach. We determine the angle sequences of the polygon and the length of the edges from the given circular embedding of the two trees. Lemmas 4.2.1 and 4.2.2 are used to determine the angle sequences and the edge lengths are taken to be the degrees of the corresponding nodes in the graph.

The following algorithm runs on a shared memory SIMD machine with p processors P_1, P_2, ..., P_p where $1 \leq p \leq n$. The input to the algorithm is the circle embedding of the visibility graph (an array of adjacency lists). The coordinates of the vertices of the orthogonal polygon (whose visibility graph is the input) constitute the output.

par-vis-realiz

(1) Subdivide the input sequence into p subsequences each of size l(length of the input list) / p and assign a processor to each subsequence.

(2) For i = 1 to p do in parallel

P_i scans its associated subsequence for the beginning of the adjacency list of any element and if it finds one it stores the corresponding array index in the shared memory so that further accesses to the adjacency lists of the elements can be done in constant time. It also stores the element to whose adjacency list the element in the beginning of its associated

subsequence belongs to

end for

(3) For i = 1 to p do in parallel

For k := 1 to n/p do

P_i finds the degree of the node (i-1)n/p+k by using the array indices found in step 2 P_i stores the degree of node (i-1)n/p+k as the length of the edge (i-1)n/p+k in the shared memory

end for

end for

(4) For i = 1 to p do in parallel

For k = 1 to n/p do

P_i initialises d[(i-1)n/p+k] = 3 and

path[(i-1)n/p+k] = 0

end for

end for

(5) For i = 1 to p do in parallel

P_i processes the subsequence associated with it

While scanning the adjacency list of node a if it encounters a node b such that b = a + 2 (mod n) then it sets d[a] = 1

If it encounter nodes b and b + 2(mod n) in the adjacency list of node a then it sets d[b] = 2 and path[b] = a

end for

(6) For i = 1 to p do in parallel

(6 1) P_i determines the angle sequence of the polygonal chain

from edge (i-1)n/p+1 to i(n/p) as follows

For each k in this range

If d[k] = 1 then the angle sequence is cc

If d[k] = 3 then the angle sequence is rr

else if node k + 1 projects across edge (k path(k))

then the angle sequence is cr

else the angle sequence is rc

(6 2) P_i determines whether edges (i-1)n/p+2 to i(n/p) are top bottom left or right assuming edge (i-1)n/p+1 to be a bottom or a left edge and using the angle sequence found above

(6 3) P_i determines the coordinates of the vertices of the edges (i-1)n/p+1 to i(n/p) taking the coordinates of the beginning vertex in the chain of edges to be (0 0) using the angle sequences found in step 6 1 and the lengths of the edges found in step 3

end for

(7) Assign n/p edges to each processor

For i = 0 to log p - 1 do

For k = 0 to n - 2^{i+1} in steps of 2^{i+1} do in parallel

P_{k+2^i+1} tests the beginning edge e_b in its associated set of edges and the last edge e_l in the set of edges associated to processor P_{k+2^i} Let e_l be a vertical edge and e_b be a horizontal bottom edge going right

Let e_l = [(x_1 y_1) (x_1 y_2)] e_b = [(x_2 y_3) (x_3 y_3)]

Case:

i e_1 is a left edge going up and the angle between the edges is conve A mirror image w r t $y = y_2$ is to be taken (fig 4 5a)

ii e_1 is a left edge going up and the angle between e_1 and e_b is refle A mirror image w r t) $_1$ is to be taken (fig 4 5b)

iii e_1 is a left edge going down and the angle between the edges is refle A mirror image w r t $y = y_2$ and with respect to =) $_1$ is to be taken (fig 4 5c)

iv e_1 is a left edge going down and the angle between the edges is conve) No mirror image need be taken

Similarly the other cases can be dealt with

For j := k + 2^i + 1 to l + 2^{i+1} do in parallel

P_j translates the coordinates of the vertices of the edges in its associated sequence Then it takes the mirror images of the vertices if needed as determined above

end for

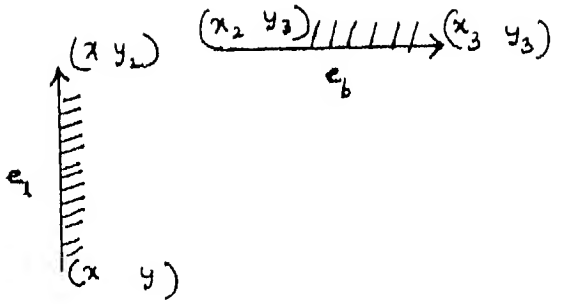
end for

end for

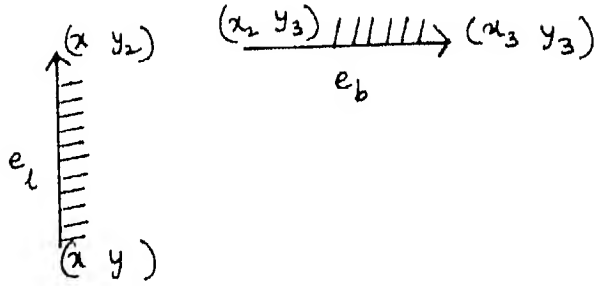
4 5 The Analysis:

Step 1 takes constant time

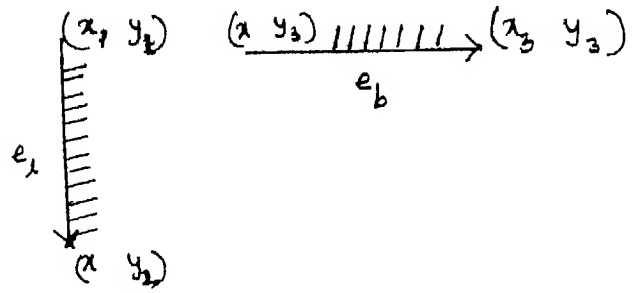
Steps 2 and 5 take $c_1 n/p$ and $c_2 n/p$ time respectively where c_1 and c_2 are constants


(x y2)
(x2 y3)
(x3 y3)
eb
el
(x y)

(a)


(x y2)
(x2 y3)
(x3 y3)
eb
el
(x y)

(b)


(x1 y1)
(x y3)
(x3 y3)
eb
el
(x y2)

(c)

Fig 4 5

Step 3 takes $c_3(n/p)$ time

Steps 4 and 6 take $c_4 n/p$ and $c_5 n/p$ tim

and c_5 are constants (In step 6 1 to

across edge (1 path(1)) it takes cor

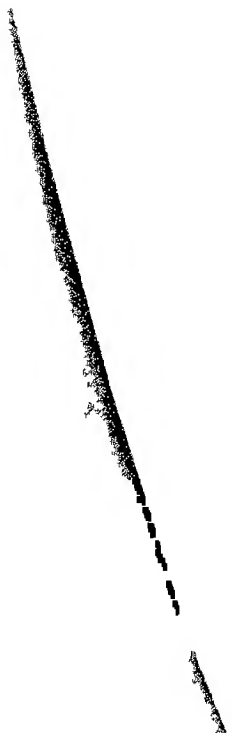
4 2 2)

Step 7 takes $c_6(\log p)n/p$ time where c

processor takes a constant time to tr

mirror image of the coordinates of one v

$$t(n) = c_1 n/p + c_2 n/p + c_3 n/p + c_4 n/p +$$
$$= O((\log p)n/p)$$



In the above algorithm taking p = n / log n we get an optimal parallel algorithm as follows In step 7 we can do the merging by doing a prefix computation of the amount of translation to be done(similarly the necessity of taking mirror images) Hence this step can be done in O(log n) time using O(n / log n) processors Therefore the time complexity of the algorithm is O(log n)

Another method to determine the coordinates of the vertices of the polygon is as follows The direction in which an edge goes changes when two convex or reflex angles occur together We note the occurence of such a sequence by -1 and the occurences of other angle sequences by +1 and then do a prefix multiplication on these Then multiplying the numbers so obtained by the edge lengths we get the relative lengths of the edges Doing a prefix sum computation on these gives the exact coodinates of the vertices Using O(n / log n) processors this can be done in O(log n) time

CHAPTER 5

CONCLUSIONS

In this thesis parallel algorithms for certain problems i Computational Geometry have been presented The shortest lin segment from which a given convex polygon is weakly external visible is computed in $O(n^e \log n)$ time using $O(n^{1-e})$ CREW PRA processors The realization of the orthogonal polygon given th circular embedding of the two visibility trees of the polygon c be done in $O((n/p) \log p)$ time using $O(p)$ CREW PRAM processor The linear programming problem the minimum spanning circ problem of a set of points lines and a set of line segments ha been solved in $O(n^e)$ time using $O(n^{1-e})$ CREW PRAM processors

Several problems remain open One is the problem of findi an efficient parallel algorithm for the computation of th shortest line segment from which a simple polygon is weak externally visible Another is to find an optimal parall algorithm for the reconstruction proble of a simple polygon giv its visibility graph

REFERENCES

[AG85] Aggarwal A Chagelle B et al Parallel Computational Geometry Proc 26th FOCS 1985 pp 468-477

[AC87] Atallah M Cole R Goodrich M Cascading Divide and Conquer A technique for designing parallel algorithms Proc 28th FOCS 1987 pp 151 160

[AC89] Atallah M J Chen D J An Optimal Parallel Algorithm for the Visibility of a Simple Polygon from a Point 5th ACM Symp on Computational Geometry 1989 pp 114 123

[AG86] Atallah M J Goodrich M T Efficient Plane Sweeping in Parallel Proc 2nd Symp on Computational Geometry 1986 pp 216 225

[AJ85] Akl S G Parallel Sorting Algorithms Academic Pres Inc 1985

[AT81] Avis D Toussaint G T An Optimal Algorithm for Determining the Visibility of a Polygon from an Edge IEEE Trans Computers C 30 12 1981 pp 910 914

[BB90] Bhattacharya B K Toussaint G T Mukhopadhyay A A Linear time Algorithm for Computing the Shortest Line Segment from which a Polygon is Weakly External Visible TRCS 90-99 Technical Report IIT Kanpur

[BM90] Bhattacharya B K Mukhopadhyay A A Linear time Algorithm for the Smallest Intersection Radius Problem TRCS 90 98 Technical Report IIT Kanpur

[BS90] Bertola i P Sal a S A Parallel Algorithm for Visibility Problem from a Point J Parallel and Distributed Computing 9 1990 pp 11 15

[BT89] Bhattacharya B T Toussaint G T Computing Minimal Sets of External Visiblity Tech Report McGill Univ , 1989

[BT90] Bhattacharya B K Toussaint G T Computing Shortest Traversals Tech Report SOCS 90 6 McGill University April 1990

[CH8_] Chagelle B A Theorem on Polygon Cutting with Applications Proc 23th FOCS 1982 pp 339 349

[DM86] Dyer M E On a Multidimensional Search Technique and its Applications to the Euclidean one center problem SIAM J Comput 15 1986 pp 725 738

[D 88] Dehne F Sack J R A Survey of Parallel Computational Geometry Algorithms LNCS 342 pp 73 88

[E 90] Everett H Corneil D G Recognising Visibility graphs of Spiral Polygons J Algorithms 11 1990 pp 1 26

[EG87] ElGindy H Goodrich M T Parallel Algorithms for Shortest path problems in Polygons MS CIS 87 20 Technical Report Univ of Pennsylvnia

[EH86] ElGindy H A Parallel Algorithm for the Shortest path problem in Monotone Polygons MS CIS 86 49 Tech Rep Univ of Pennsylvania

[GS90] Goodrich M T Shauck S B Guha S Parallel Methods for Visibility and Shortest Path Problems in Simple Polygons ACM Symposium on Computational Geometry 1990 pp 73 81

[HJ90] Huang J H Optimal Parallel Merging and Sorting Algorithms using √n Processors without Memory Contention Parallel Computing 14 1990 pp 89 97

[JM90] Jadhav S Mukhopadhyay A Bhattacharya B K A Linear time Algorithm for the Smallest Intersection Radius of a set of Line Segments in the plane TRCS 90 106 Technical Report IIT Kanpur

[KC83] Kruskal C P Searching Merging and Sorting in Parallel Computation ICEE Trans on Computere C3 (10) 1983 pp942 946

[KR85] Kruskal C P Rudolph L Snir M The Power of Parallel Prefix IFEE Trans on Computers C 34 1985 pp 965-968

[KR88] Karp Ramachandran A Survey of Parallel Algorithms for Shared Memory Machines UCB/CSD88/408 Tech Report Univ of California Berkley

[LF80] Ladner R E Fisher M J Parallel Prefix Computation J ACM 1980 pp 831 838

[LP84] Lee D T Preparata F P Computational Geometry - A Survey IEFE Trans on Computers Vol C-33 12 1984

pp 1072 1101

[MNO84] Megiddo N Linear time Algorithm for Linear Programming in R^3 and Related Problems SIAM J Comput 12 1983 pp 759 776

[O'R87] O Rourke J Art Gallery Theorems and Applications Oxford University Press 1987

[OW88] Overmars M H Wood D On Rectangular Visibility J Algorithms 9 1988 pp 372 390

[P J] Preparata F P Shamos M I Computational Geometry An Introduction Springer Verlag New York 1985

[QM87] Quinn M J Designing Efficient Algorithms for Parallel Computers McGraw Hill International Editions 1987

[RC88] Richard Cole Parallel Merge Sort SIAM J Comput 17 1988 pp 770 785

[SM75] Shamos M I Geometric Complexity Proc 7th ACM Sympos Theory of Comput 1975 pp 224 233

[SS89] Sack J R Suri S An Optimal Algorithm for Detecting Weak Visibility of a Polygon Proc 5th STACS 1988 pp 312-321

[VL75] Parallelism in Comparision Problems SIAM J Comput 4 3, 1975 pp 348 355

[WT87] Wang C A Tsin Y H An O(log n) time Parallel Algorithm for Triangulating a set of Points in the plane IPL 25 1987 pp 55 60

[YC87] Yap C K What can be parallelised in Computational Geometry LNCS 269 pp 184 195